DYNAMIC SIMULATION OF CHEMICAL PLANT

by

JOEL EMENIKE OGBONDA

PhD

The University of Aston in Birmingham

June 1987

The University of Aston in Birmingham

DYNAMIC SIMULATION OF CHEMICAL PLANT.

Joel Emenike Ogbonda

Doctor of Philosophy

1987

SUMMARY

This thesis describes the design and implementation of a new dynamic simulator called DASP. It is a computer program package written in standard Fortran 77 for the dynamic analysis and simulation of chemical plants. Its main uses include the investigation of a plant's response to disturbances, the determination of the optimal ranges and sensitivities of controller settings and the simulation of the startup and shutdown of chemical plants.

The design and structure of the program and a number of features incorporated into it combine to make DASP an effective tool for dynamic simulation. It is an equation-oriented dynamic simulator but the model equations describing the user's problem are generated from in-built model equation library. A combination of the structuring of the model subroutines, the concept of a unit module, and the use of the connection matrix of the problem given by the user have been exploited to achieve this objective. The Executive program has a structure similar to that of a CSSL-type simulator.

DASP solves a system of differential equations coupled to nonlinear algebraic equations using an advanced mixed equation solver. The strategy used in formulating the model equations makes it possible to obtain the steady state solution of the problem using the same model equations. DASP can handle state and time events in an efficient way and this includes the modification of the flowsheet.

DASP is highly portable and this has been demonstrated by running it on a number of computers with only trivial modifications. The program runs on a microcomputer with 640 kByte of memory. It is a semi-interactive program, with the bulk of all input data given in pre-prepared data files while communication with the user is via an interactive terminal. Using the features in-built in the package, the user can view or modify the values of any input data, variables and parameters in the model, and modify the structure of the flowsheet of the problem during a simulation session. The program has been demonstrated and verified using a number of example problems.

Key words:

Dynamic Simulation;
Differential-Algebraic Equations;
Computer-aided design;
Equation-oriented Approach;
Simulation;
Process Dynamic Simulator.

DEDICATION

This thesis is dedicated to my late uncle, Mr Gabriel Ogbondamati Ogbonda, who watered the seed of my success when it was most needed. He saw the seed germinate and grow but he passed away before it could bear fruits. His fatherly advice has always guided me. He will always be remembered by me. May his soul rest in perfect peace.

ACKNOWLEDGEMENTS

May I express my gratitude and appreciation of the contribution and support in various ways of the following:

Dr John Fletcher, who supervised the work and read the manuscripts ; Dr Brian Gay, who offered a critical advice on the write-up and gave permision for the use of the Mackintosh Laboratory to type the thesis; Dr APH Jordan, whose interest, advice and discussions were very valuable.

My wife, Irene, who offered her moral support and understanding in spite of the occasional "neglect" she had to put up with; Dr Emma Ezegwu, Dr Okeh Igwe, Dr Mike Vernon and Mr Chris Ani whose interest and moral support kept me going; all my colleagues at the computation laboratory of the Department of Chemical Engineering for the mutual advice offered to one another and Miss Tracy Knight for typing the thesis.

Finally, I must express my thanks to the Federal Government of Nigeria for providing the financial support to carry out this work.

LIST OF CONTENTS

		PAGE NO
TITLE		1
SUMMARY		2
DEDICATION		3
ACKNOWLEDGEMENTS		4
LIST OF CONTENTS		5
LIST OF TABLES		11
LIST OF FIGURES		11
LIST OF PUBLICATIONS		13
CHAPTER 1 - PROCESS DYNAMIC SIMULATION		14
1.1	Simulation	14
1.2	Usefulness of Dynamic Simulation Studies	15
1.3	Developments in Dynamic Simulation	16
1.3.1	The Sequential Modular Method	17
1.3.2	The Equation-Oriented Method	18
1.4	Problems with Dynamic Simulation Studies	19
1.4.1	Difficulties of Modelling and Simulation	19
1.4.2	Stiffness of the Model Equations	19
1.4.3	Event Processing	20
1.4.4	Initialization of DAEs	22
1.5	Tools For A New Dynamic Simulator	22
1.6	Organization of the Thesis	25
CHAPTER 2 - BASIC DESIGN STRUCTURES		27
2.1	Introduction	27
2.2	The Sequential Modular Approach	30
2.2.1	Application to Dynamic Simulation	31

2.2.2 Formulation of the Model Equations 32
2.2.3 Coupled and Uncoupled Solution Methods 32
2.2.4 Limitations of the Sequential Modular Method 33
2.3 The Equation-oriented (E-O) Approach 34
2.3.1 Problem with the Equation-oriented Approach 35
2.3.2 Application to Dynamic Simulation 37
2.3.3 Model Formulation Strategies 37
2.3.4 Modularly-Organized E-O Simulation 41
2.4 Simultaneous Modular Method 42
2.4.1 Approaches to Simultaneous Modular Method 43
2.5 Discussion 43

CHAPTER 3 - NUMERICAL SOLUTION METHODS 46
3.1 Introduction 46
3.2 The Numerical Solution of ODEs 47
3.2.1 Stiffness of Ordinary Differential Equations 49
3.2.2 A Brief Description of the BDF Methods 51
3.3 Analysis of Approaches for Solving DAEs 52
3.3.1 Solvability of DAEs using BDF methods 56
3.4 The Handling of Discontinuities 60
3.5 Solution of Nonlinear Algebraic Equations 63
3.6 Conclusion 66

CHAPTER 4 - FEATURES OF DYNAMIC SIMULATORS 68
4.1 Introduction 68
4.2 A Description of ACES 68
4.2.1 Main Features of ACES 68
4.2.2 Defficiencies of ACES 69
4.3 The CSSLs 71
4.4 DPS 73

4.5	SPEEDUP	75
4.6	Requirements for DASP	76
4.7	The Computer Environment	79
4.7.1	ICL PERQ	80
4.8	Conclusion	82
CHAPTER 5 - THE DESIGN OF DASP		83
5.1	The Design Philosophy	83
5.2	The Main Features of DASP	86
5.3	The Structure of DASP	88
5.4	The Initial Region	90
5.5	The Dynamic Region	93
5.5.1	Steady State Simulation Section	94
5.5.2	Dynamic Simulation Section	94
5.5.3	Perturbation Section	95
5.5.4	Event Processing Section	95
5.6	The Terminal Region	96
5.6.1	End Simulation / Rerun Section	96
5.6.2	Error Analysis Section	97
5.7	Conclusion	98
CHAPTER 6 - EQUATION GENERATION		99
6.1	Introduction	99
6.2	DASP's Basic Terminology	100
6.2.1	Variables and Parameters	100
6.2.2	Module or Model Routine	100
6.2.3	Unit Module	100
6.2.4	Staged Module	101
6.2.5	Sectionalized Module	101
6.2.6	Unit	102

6.2.7 Stream 102
6.3 The Model Routine 102
6.3.1 Description of Common Block Variables 103
6.3.2 Description of Argument Lists of Subroutine 104
6.3.3 Initialization Section 105
6.3.4 Function Evaluation Section 108
6.3.5 Jacobian Evaluation Section 108
6.3.6 Nonzeros of Sparse Matrix Setup Section 109
6.3.7 Output Section 109
6.3.8 Event Code Section 109
6.3.9 Terminal Section 109
6.4 Transfer of Variables Between Modules 110
6.5 Assembling of Equations from Modules 112
6.6 The USRSUB Option 114
6.7 Conclusion 115

CHAPTER 7 - THE EQUATION SOLVING PACKAGE 116
7.1 Introduction 116
7.2 Description of DASSL 117
7.3 Adaptation of DASSL 120
7.3.1 Modification of Argument List of Subroutines 120
7.3.2 Replacement of the Error Routine, XERRWV 121
7.3.3 The Incorporation of the Sparse Matrix Option 121
7.3.4 Implementation of Event Processing Option 121
7.3.5 Change of Execution Logic of the Integrator 121
7.3.6 New Approach to Initialization of DAEs 122
7.4 The Nonlinear Equation Solvers 123
7.4.1 The Newton-Raphson Routine, NRSUB 123
7.4.2 The Modified Broyden Routine, BROYDN 126
7.5 The Service Routines 127

7.5.1 Function Evaluation Routine, GETFUN 127
7.5.2 Jacobian Evaluation Routine, GETJAC 128
7.5.3 Linear Equation Solvers 130
7.6 Conclusion 130

CHAPTER 8 - THE EVENT PROCESSING PACKAGE 131
8.1 Introduction 131
8.2 Event Detection 133
8.3 Event Time Determination 134
8.4 The Event Logic 135
8.4.1 Exit 136
8.4.2 Change Variable / Parameter Values 136
8.4.3 Disconnect a Unit from the Flowsheet 137
8.4.4 Reconnect a Unit to the Flowsheet 138
8.4.5 Call a Module Event Section 141
8.4.6 Call a User-written Event Code via EVCODE 141
8.4.7 Call a User-written Event Code via "USRSUB" 142
8.5 The Upset Functions 143
8.6 The MODIFY Routine 144
8.7 Discussion on Event Processing Package 151

CHAPTER 9 - DEMONSTRATION OF DASP 153
9.1 Introduction 153
9.2 Example Type I - Stiff Differential Equations 153
9.2.1 Description of Example Type I Problems 154
9.2.2 Solution of Example Type I problems 156
9.3 Example Type II - Nonlinear Algebraic Equations 157
9.3.1 Description of Example Type II Problems 158
9.3.2 Solution of Example Type II Problems 158
9.4 Example Type III - DAE Systems using USRSUB Option 159

9.4.1 Description of Example Type III Problems 159
9.4.2 Solution of Example Type III Problems 162
9.5 Example Type IV - DAE Systems using DASP Modules 163
9.5.1 Description of Methanol Mixer Problem 163
9.5.2 Solution of Methanol Mixer Problem 166
9.5.3 Description of Mixer-Splitter Problem 168
9.5.2 Solution of Mixer-Splitter Problem 170
9.6 Example Type V - Special Features of DASP 171
9.6.1 Demonstration using USRSUB Option 171
9.6.2 Demonstration using Models from DASP 173
9.7 Conclusion 175

CHAPTER 10 - DISCUSSION AND CONCLUSION 176
10.1 Introduction 176
10.2 The Structure of DASP 177
10.3 The Solution of DAEs 178
10.4 The Initialization of DAEs 178
10.5 Organization of Equations in Modules 179
10.6 Event Processing 179
10.7 Special Features of DASP 180
10.8 Demonstration of DASP 180
10.9 Limitations and Future Work 181

APPENDIX A - RESULTS OF EXAMPLE PROBLEMS 182
APPENDIX B - DASP MINI-MANUAL 195
APPENDIX C - AN ALTERNATIVE IMPLEMENTATION OF DASP 290
REFERENCES 302

LIST OF TABLES

PAGE NO

Table 8.1 - Incidence Matrix of Figure 8.2 140

Table 8.2 - Connection Matrix of Figure 8.2 140

Table 9.1 - Results of Example Type I Problems 157

Table 9.2 - Description of Block Diagram in Figure 9.4 175

LIST OF FIGURES

Figure 1.1 - Flowsheet of a Water Heater 20

Figure 2.1 - Variable Types in a System 28

Figure 2.2 - Flowsheet of a Plant Illustrating Bidirectional Flow 34

Figure 3.1 - Discontinuity in the Derivative of the State Variable 61

Figure 4.1 - The Structure of a Typical CSSL 72

Figure 5.1 - Overall Structure of DASP 91

Figure 6.1 - An Example of Translation of a Flowsheet to Blockdiagram 101

Figure 6.2 - Connection Blocks of Modules 111

Figure 6.3 - Block Diagram of a Process 111

Figure 6.4 - Arrangement of residuals of Modules in FUNC(*) 114

Figure 7.1 - Structure of a Jacobian Matrix 128

Figure 8.1 - Structure of the Event Processing Package 132

Figure 8.2 - A Block Diagram of a Flowsheet 139

Figure 8.3 - Connection Blocks of Units in Figure 8.2 140

Figure 9.1 - Process Flowsheet of Problem No. 8 160

Figure 9.2 - Plot of Temperature and Concentration versus Time 163

Figure 9.3 - Flowsheet of Methanol Mixer Problem 165

Figure 9.4 - Block Diagram of Methanol Mixer Problem 165

Figure 9.5 - Plot of Liquid Height and Concentration versus Time 168

Figure 9.6 - Mixer Splitter System Flowsheet 168

Figure 9.7 - Block Diagram of Mixer Splitter System 169

Figure 9.8 - Plot of Butane Concentration versus Time 171

Figure 9.9 - Plot of Concentration versus Time (Event Option) 172

Figure 9.10 - Plot of Methanol Concentration versus Time (Event) 174

Figure 9.11 - Plot of Methanol Concentration versus Time (Event + Upset) 174

LIST OF PUBLICATIONS

1. Fletcher JP and Ogbonda JE (1985), "The Development of a Dynamic Simulation Package for Chemical Processes", Design '85, Conference on Computers and Design in the Process Industries, Organised by Midland Branch of IChemE, University of Aston in Birmingham, September 11-12, 1985.

2. Fletcher JP and Ogbonda JE (1987), "A Modular Equation-Oriented Approach to Dynamic Simulation of Chemical Processes", Proceedings of XVIII Congress on The Use of Computers in Chemical Engineering, CEF '87, Giardini Naxos, Sicily, Italy, April 26-30, 1987.

CHAPTER 1

PROCESS DYNAMIC SIMULATION

1.1 SIMULATION

Simulation has become an accepted tool in the design and analysis of chemical plants. Today it is difficult to imagine any new large-scale chemical plant being built without first being simulated on a digital computer. Simulation starts with the construction of the mathematical model of the physical and chemical phenomena taking place in the chemical plant. The variables in this model represent particular characteristics or states of the process. The model is then experimented on by solving the model equations to generate values of the variables which satisfy the model constraints.

We can distinguish between steady state and dynamic simulations. In the steady state, the model equations usually consist of nonlinear algebraic and differential equations and the variable values do not vary with time. Steady state simulation is mostly applied to mass and energy balance calculations, costing and preliminary optimization of chemical plants. Dynamic simulation models consist of differential and algebraic equations and the variables are time-dependent. It can be used to check steady state operations under different conditions. Simulation can further be classified as stochastic if the model relies on the laws of probability and uses mean values, or deterministic if the variables are single-valued. In continuous simulation the variables can take any values within a given interval while in discrete simulation there is a discrete jump in the values of the variables. This research is concerned with the dynamic simulation of continuous deterministic chemical processes, in which the steady state is treated as a special case of dynamic simulation.

1.2 USEFULNESS OF DYNAMIC SIMULATION STUDIES

The process design of a chemical plant can go through the synthesis, analysis and optimization stages. In synthesis, the necessary pieces of equipment and their order of connection in the flowsheet are synthesized. The analysis stage covers both steady state and dynamic simulations. During optimization, the designer determines the optimum values of the flowsheet variables and parameters. In each of the above stages, a designer would like to know answers to questions of the type "what would happen if .....". Some of these answers can be provided by dynamic simulation studies and will play a vital role in decision making for improvement in the design, or the testing of new designs before they are built. The following are some of the main areas of application of dynamic simulation studies:

1. Startup and Shutdown of Chemical Processes

Information about the stability of the processs during start-up and shut-down can help process engineers evolve better start-up and shut-down procedures.

2. Effects of Parameter Changes

Investigation of the effects of feedstock changes on the process variables can be carried out. This could help to determine whether the plant can cope with feedstocks from different sources which may have different concentrations.

3. Control System Design

The optimum ranges and sensitivities of controller settings can be determined. In order to specify the instrumentation of a process, the designers have to rely on their past experience with similar systems, which usually results in using large overdesign factors in order to make the control scheme safe. Dynamic simulation can be used to check different control schemes by incorporating control loops and control algorithms into the model and then tuning the controller settings to give optimum process response to a given perturbation. Also with the high cost of instrumentation, the model can be used to

identify which parameters have to be measured and those which do not. Once the desired control scheme has been selected, the final plant sizing is performed based on dynamic as well as steady state considerations.

4. Operating Strategies for Multiproduct Plants

Operating policies for multiproduct plants can be devised for efficient changeovers.

5. Operator Training

Dynamic simulation can be used to train plant operators by simulating the model of the plant. In this way they can familiarize themselves with the operation of the plant before the actual running of the plant.

1.3 DEVELOPMENTS IN DYNAMIC SIMULATION

Starting from the early 1950's, electrical engineers conducted dynamic simulation studies using analog computers, in which voltage is used to represent the variables and parameters of the system under study as well as time. The user is required to set up the electrical circuits of the process and observe the voltage changes which represented changes in the values of the process variables and parameters. This is a labourious and time-consuming process.

With the advent of digital computers, continuous system simulation languages (CSSL) emerged, which were designed to duplicate the functions of analog computers by solving the ordinary differential equations of the process model (Brennan and Linebarger, 1964). Later, several modified versions of these languages were developed and today they can compete to some extent with present-day dynamic simulators. Examples include ACSL (Mitchell, 1978), DARE-P (Wait and Clarke, 1978). Hartzog et al. (1982) showed how CSSL-type simulators can be applied to the dynamic simulation of a chemical process.

However, as Thambynayagom et al. (1981), Cameron (1981), and Joglekar and Reklaitis (1984) have stated, CSSL-type simulators do not cater for the needs of chemical process engineers and this has limited their use in the chemical industry. For example, they offer little in the way of a library of unit operation model routines. This means each problem has to be modelled and coded by the user. Secondly, they have poor error reporting facilities, which makes diagnosis very difficult in the event of a failure. Moreover, chemical engineers prefer modular simulations.

1.3.1 THE SEQUENTIAL MODULAR METHOD

The early 1970's saw the development of many process dynamic simulators specifically for the chemical industries. These programs use the sequential modular architecture, following their steady state counterparts. Several assumptions are made which allow for easier and more efficient solutions. These include:

1. The variables associated with the units (design variables) should always be defined and should not be treated as unknowns. The same assumptions are made for variables associated with streams entering the plant from outside (feed streams).

2. The information flow in the mathematical model should coincide with the material flow in the chemical plant.

With these assumptions, unit operation models were coded as Fortran subroutines. These modules, as they are called, are structured so that given the input variables, model parameters, and the output variables at time t, the derivatives of the output variables are calculated. This is called derivative evaluation. The module then calls the integration algorithm to integrate the equations from time t to time t+h, where h is the integration time step. In a simulation session, the executive routine of the simulator calls these unit modules in a predetermined sequence with the values of the input variables, parameters

and output variables at time t to calculate the values of the output variables at time t+h. This process is repeated until the final integration time is reached. Examples of simulators of this design include DYFLO (Franks, 1972), DISCO (Briggs, 1974), DYNSYS (Barney, 1975), DYNSYL (Patterson and Rozsa, 1980). Dynamic simulation studies using this approach run into difficulties with design, optimization and recycle process calculations (Westerberg et al, 1979; Shacham et al, 1982) and the solution of differential equations coupled to nonlinear algebraic equations (Cameron, 1981). Further discussion of the sequential approach is presented in Section 2.2 of Chapter 2 in greater detail.

1.3.2 THE EQUATION-ORIENTED METHOD

An alternative method to sequential modular simulation exists which can conveniently overcome all the problems associated with it. This is the equation-oriented approach, whose main feature is that all the model equations are solved simultaneously during the integration process. In a fully equation-oriented approach, all the model equations are provided by the user as a large system of equations and the simulator just act as an equation solver. In a variant of the equation-oriented approach, the model equations are generated from in-built model equation library called modules. This approach is called the modularly organized equation-oriented approach. This is now preferred as it relieves the user of the effort of writing the model equations for his problem. Examples of dynamic simulators using the equation-oriented approach include DPS (Thambynayagom et al, 1981), SPEED-UP (Perkins and Sargent, 1982), FLOWSIM V4.0 (Babcock, 1982), BOSS (Joglekar and Reklaitis, 1984). The equation-oriented approach is beseiged with various implementation problems which need to be solved. A more detailed discussion is presented in Section 2.3 of Chapter 2.

1.4 PROBLEMS WITH DYNAMIC SIMULATION STUDIES

Compared to steady state simulators the development of dynamic simulation packages have been very slow. This has been due to the many problems which had to be overcome. Steady state and dynamic simulation merge at several levels. Both yield a set of partial differential and algebraic equations and may require similar numerical solution techniques (it is possible to solve a steady state model using dynamic simulation algorithms). Steady state simulation can provide initial values needed for dynamic simulation. The two can also share the same physical property package for calculating thermodynamic constants and variables used in the calculations. However, dynamic models have several characteristics which are different from steady state models and thus require special techniques for their solution. The problem to be solved in dynamic simulation can be described as the solution of a system of differential equations coupled to a system of algebraic equations, normally called Differential-Algebraic Equations or DAEs for short. The problem may include subroutines of procedures (e.g. associated thermodynamic models), events (described in section 1.4.3 below) and analysis of the result in order to provide answers to the questions posed by the problem. A few of the problems usually encountered in dynamic simulation studies are briefly discussed below.

1.4.1 DIFFICULTIES OF MODELLING AND SOLUTION

Modelling the dynamic behaviour of chemical processes can be very difficult and requires a thorough understanding of the process. Although certain simplifying assumptions are usually made, the resulting model equations can be very difficult to solve using available numerical algorithms.

1.4.2 STIFFNESS OF THE MODEL EQUATIONS

One of the characteristics of dynamic models of chemical plants is that the systems of equations have time constants which differ in orders of magnitude. This causes the

numerical methods which were hitherto used for their solution (known as explicit methods) to use very small time steps and hence large computer time. Today this problem, known as stiffness, can be overcome using new methods designed for these types of problems (Gear, 1971a ; Carnahan and Wilkes, 1980).

1.4.3 EVENT PROCESSING

In dynamic simulation, state and time events do occur, especially during start-up and shut-down of chemical processes. To understand these events, consider the simple flowsheet shown in fig. 1.1, which consists of two tanks, **T1** and **T2** and three valves, **V1**, **V2** and **V3.**

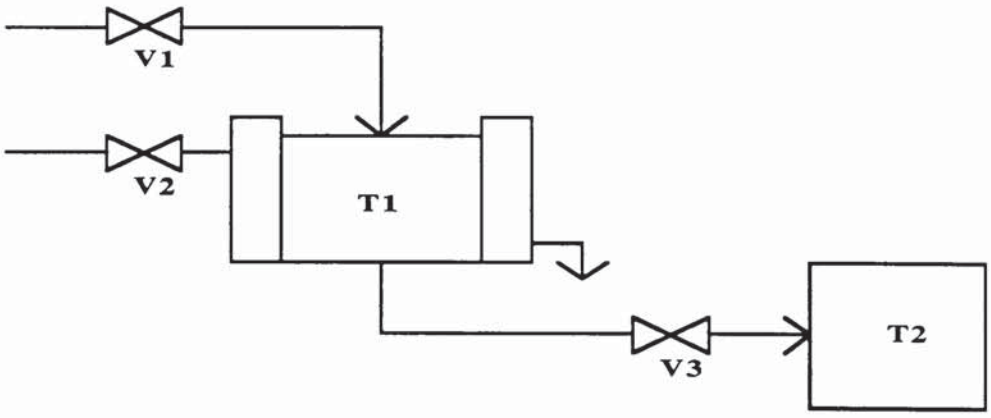


Fig. 1.1 Flowsheet of a water heater.

It may be necessary to operate this flowsheet in the following sequence, assuming that at the beginning (time TM1), all the tanks are empty and all the valves are closed:

a. Event 1

At time TM1, open valve **V1** to allow water to flow into tank **T1** up to the level LEV1.

b. Event 2

When the water level has reached the level LEV1 (i.e. at time TM2), close valve **V1** and

open valve **V2** to heat the water to a specified temperature TEMP1 (i.e. at time TM3).

c. Event 3

When the temperature has reached TEMP1, close valve **V2** and open valve **V3** to allow the hot water to flow into tank **T2**.

In the above operation, event no. 1 is a time event since the time of occurence is known a priori. On the other hand, event no. 2 and 3 are state events as their times of occurence are determined by constraints on other units and variables. The detection of an event and calculation of the time of its occurence are the problems to be tackled.

Firstly, a point constraint results when, as a result of switching from one valve to another, the model equation to be solved changes. The problems in state event processing include the detection and the calculation of the time of its occurence. If this time is computed successfully, the state variable values are then calculated at this time. Integration is later restarted from this time. This procedure is commonly known as integration over a discontinuity. Various attempts have been made to solve this problem (Carvar, 1978; Ellison, 1981). For example, Kuru (1981) described a method whereby once a discontinuity is detected, the current step of the integration is abandoned and an equation defining the point of discontinuity is added to the set of equations being solved and the stepsize of integration is added as a new variable to be calculated. While this may work for a case of one event, it is not clear how multiple events can be handled. Also this procedure not only increases the number of equations and variables but require modifications in the sizes of the state vectors and other work arrays used for the solution.

The second problem in event processing is handling of latency. Latency results when the values of the variables in a unit either do no change at all for dynamically inactive units or do not change significantly for dynamically less active units. For

example, in Event 1 above, valves **V2** and **V3** and tank **T2** were inactive. A method need to be devised to "remove" or disconnect the dynamically inactive units during simulation of a flowsheet. This will greatly reduce the unnecessary computations done on these units. Kuru and Westerberg (1985) described a method of handling dynamically less active units when they are detected during the integration process, in which these units are detected and certain computational procedures are bypassed for these units. They reported substantial savings in total computation time.

A third problem in event processing is the restarting of the integration after an event has been processed and the relevant event code executed. This problem is most acute for those numerical integration methods which need the previous values of the variables or their derivatives to advance to the next time step. These include the Backward Differentiation Formulas (BDF) discussed in Chapter 3. One solution is to use a first order method of integration and a small steplength for the first step, and then increase the order and steplength afterwards.

1.4.4 INITIALIZATION OF DAES

A scheme needs to be devised to provide a consistent initial values for both the variables and derivatives in the model. This is still an unsolved problem in dynamic simulation as is the case of providing the initial guesses for all the variables in steady state simulation within an equation-oriented environment (Locke, 1981).

1.5 TOOLS FOR A NEW DYNAMIC SIMULATOR

The last few years have seen tremenduous increase in the number of tools, which can be harnessed by developers of dynamic simulation programs to overcome most of the problems mentioned in Section 1.4. These new tools include:

1. New numerical algorithms and methods which can handle problems such as the solution of general differential-algebraic equations, stiff systems of differential equations and highly nonlinear systems of algebraic equations. These are discussed in Chapter 3.

2. Cheap and powerful microcomputers, with large memory capacity, thus eliminating the computer memory problems of earlier designs on microcomputers.

3. The use of the equation-oriented approach, in which the model equations are gathered together from in-built model equation library and solved simultaneously. This offers flexibility in terms of formulation of the model equations for solution, selection of design variables and optimization which were some of the main problems with the sequential modular approach. These are discussed in detail in Chapter 2.

4. New Chemical Engineering-oriented databases, which can be used for efficient transfer of data and information between the user- and simulator-ends of the system. The discussion of databases is beyond the scope of this work but comprehensive information can be found in Leesley and Buchmann (1980) and Tsubaki and Motard (1979)

Shacham et al (1982) pointed out that the technology needed for the development of state-of-the-art equation-oriented simulators is already available and that what is needed is a mechanism for assembling it into a working package. This is confirmed by the increasingly large number of steady state simulators being developed at different research centres and universities all over the world, as mentioned in Section 1.3. However, there are only a few general-purpose dynamic simulators harnessing these ideas, and they do not exploit them to the full. For example, SPEEDUP (Perkins and Sargent, 1982), which can be regarded as one of the most advanced of all is developed for mainframe computers and thus cannot exploit the advantages of the more user-friendly environment of microcomputers. Also ASCEND II (Kuru, 1981) and DYNAMIC QUASILIN

(Smith and Morton, 1987) use a restricted model formulation strategy as discussed in Chapter 2.

One of the reasons for the lack of application of these new methods is that these tools were independently developed by people of various disciplines without any co-ordination towards a common goal. The chemical engineer, who will use the package at the end, is best equipped to formulate the model in terms of the required equations. Numerical analysis is the task of the mathematician, who must select and develop the most appropriate set of procedures, called algorithms, to solve the equations by arithmetical and logical operations. The writing of an efficient program is best carried out by a computer scientist or programmer. However, an engineer designing a package must understand to a large extent the works of the various scientists and engineers in order to interface or integrate their work into a working and efficient simulation package.

Secondly, it is not an easy task incorporating these new developments into a working dynamic simulation package. Firstly, the design of an equation-oriented simulator executive is a very difficult task as pointed out by Perkins (1984). Secondly, the new software was all designed by workers of various backgrounds for solving general mathematical problems without regard to the particular needs of the chemical engineer. Therefore, these programs have to be adapted and modified to the requirements of the simulator designer.

It is therefore, the objective of this work to develop a mechanism for exploiting these new methods and algorithms for the purpose of designing a new process dynamic simulator, capable of running on a micro computer, with 640 KByte of main memory. This thesis describes the design of such a simulator called DASP. The thesis demonstrates that these new ideas, though from different disciplines, can be combined into a working dynamic simulator. Also, it should present a modular appearance to the user by organising the equations to be solved from an in-built model equation library,

although these equations will be solved simultaneously. This requires the development of suitable structures for organizing the equations from an in-built model equation library, a new approach to the initialization of differential-algebraic equations, a method of handling dynamically inactive units in the flowsheet and flexible and simple structure for the executive program. The design philosophy and main features of the package are described in Chapter 5.

1.6 ORGANIZATION OF THE THESIS

This thesis is divided into ten Chapters and three Appendices. The literature survey presented in the thesis is based on the issues involved in dynamic simulation rather than on the implementation details of available dynamic simulators. This is because no such implementation details or codes were available to the author. These issues are the design methodology, the numerical methods of solving the model equations and the main features of available dynamic simulators and these are discussed in separate chapters. In Chapter 2, the design structures for a general-purpose steady state simulator are surveyed and their application to dynamic simulator design explained and analysed. Chapter 3 deals with the survey of numerical methods for solving the dynamic model equations of chemical plants. It analyses the use of a class of numerical algorithms called Backward Differentiation Formulas, BDF, to solve differential-algebraic equations (DAEs) and shows what types of problems are solvable and summarizes the reasons why certain codes cannot handle some classes of DAEs. Numerical methods for handling discontinuities are also discussed as well as methods for solving nonlinear algebraic equations. In Chapter 4, the main features of some of the existing simulators using the equation-oriented approach are described and their main deficiencies and novelties evaluated. Also, the requirements of a dynamic simulator intended to satisfy the objective of this work, and the computer enviroment available for developing this work are described.

Chapters 5 through to Chapter 8 present the process dynamic simulator, DASP, which is the product of the work described in this thesis. In Chapter 5, the design philosophy, the main features and structure of the program are described. It shows how the Executive program works via interface routines and coordinates the working of the other parts of the package. Chapter 6 starts with the definition of the terminology of the program, which make it easier to understand the module concepts developed as part of the overall mechanism of gathering the equations to be solved from in-built model equation library. The equation solving package presented in Chapter 7 describes the programs that solves the model equations, both differential-algebraic systems and nonlinear algebraic equations and the method used in DASP to initialize the DAEs. Event processing is the topic of Chapter 8, which describes how state and time events are handled by the simulator. Other features of the program which enable a user to communicate with the program during simulation are explained.

In Chapter 9, examples are used to demonstrate the working of DASP, while in Chapter 10 a discussion of the work done with regard to its novelties and deficiencies is given, which concludes with recommendations for future work. Appendix A presents the results of the example problems solved in Chapter 9 while Appendix B is a mini-manual, which describes how to access the program and run it on the IBM PC AT computer. It also includes a description of how to write a new model equation routine, and the model routine library. Appendix C describes a proposal for an alternative implementation of DASP based on the experience of this first implementation. Details of the contents of these Appendices are given in each Appendix.

CHAPTER 2

BASIC DESIGN STRUCTURES

2.1 INTRODUCTION

A process dynamic simulator can be defined as a computer program package, usually constructed on certain building blocks, for the simulation of the dynamic behaviour of chemical plants. The building blocks of a computer simulation package include the models, the algorithms, the software and the user interface. The model equations form the basis for analysis. The lumped parameter modelling of the dynamic behaviour of chemical processes usually results in ordinary differential and algebraic equations of the form:

$$\mathbf{F}(\mathbf{y}',\mathbf{y},\mathbf{z},\mathbf{p},\mathbf{x},t) = \mathbf{0} \qquad ..2.1$$

where

y is the vector of the differential variables of the system,

y' is the vector of the derivatives of the differential variables,

z is the vector of the algebraic variables of the system,

p is the vector of the equipment parameters in the system,

x is the vector of the input variables to the system,

t is the vector of the time.

Equation 2.1 can be written in standard or state variable form as

$$\mathbf{y}' = \mathbf{f}(\mathbf{y},\mathbf{z},\mathbf{p},\mathbf{x},t) \qquad ..2.1a$$

$$\mathbf{0} = \mathbf{g}(\mathbf{y},\mathbf{z},\mathbf{p},\mathbf{x},t) \qquad ..2.1b$$

The variables, **y** are sometimes called output variables. Fig. 2.1 illustrates the variables of the system.

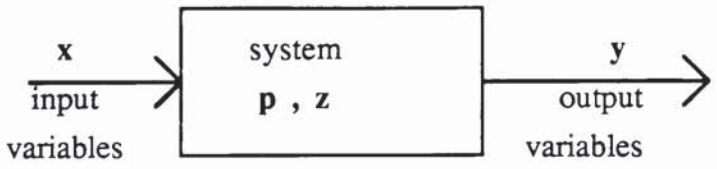


Fig 2.1 - Variable types in a system

These systems of model equations generally have the following characteristics:

a. large, in which there may be from hundreds to thousands of variables,

b. nonlinear,

c. sparse, with only a few of the variables appearing in any of the equations,

d. stiff, where the system of equations have time constants of different orders of magnitude.

These characteristics make them very difficult to solve, especially when the flowsheet of the plant has recycles, thus requiring special techniques for the solution. The equations are derived from conservation laws, rate equations, physical property relations, connection equations and design and control constraints. The mathematical modelling of chemical processes has been discussed extensively in the literature (see for example Holland and Liapis, 1983 ; Franks, 1972). Usually these equations are formulated according to a chosen solution method and converted into computer subprograms called model routines or modules.

The difficulties of solution are such that a range of techniques are required for the solution. The algorithms needed to solve these equations range from simple to complicated numerical techniques which act on the equations to produce the required numerical results. The requirements of a good algorithm include robustness, generality and efficiency in terms of the necessary storage and execution time.

The software for implementing the algorithms on a particular computer include the

operating system, data structures, a file system interface and programing languages. Good software is expected to be easy to understand, maintain and modify. The user interface provides the input language by which a user describes his problem, the reports that contain the results and the user documentation that explains how to access the system and interface with other programs. With the help of a supervisory program, the executive routine, these building blocks are structured to solve the user's problem.

There are three main structures usually adopted in the design of process dynamic simulators:

1. Sequential modular method (SEM),
2. Equation-oriented solution method (E-O),
3. Simultaneous modular approach (SIM).

These structures are discussed in detail in the next three sections. They were originally developed and applied to flowsheeting systems (Westerberg et al, 1979), which are large-scale computer programs that perform material and energy balancing, equipment sizing and costing calculations in the steady state design and optimization of chemical plants. Most of the experience in the application of these solution methods is on flowsheeting systems, from where they have been adapted to the requirements of dynamic simulation. There are some similarities in the requirements of the two types of systems, for example the same physical property calculation routines and input and output format can be used. There have been a few attempts to convert existing sequential modular flowsheeting systems so that they can do dynamic simulation as well. An example of such a system is DYNAMIC FLOWPACK II (Aylott et al, 1984). This approach is intended to reduce the cost of developing new dynamic simulation packages by making use of the huge investments in existing programs for the physico-chemical property routines and algebraic equations solving routines, and user training by retaining a familiar user interface.

Transient behaviour arises because of the inbalance between the inlet and outlet streams in a process. Moreover, dynamic simulation requires dynamic models which can be regarded as extensions to the steady state models. The solution algorithms are usually different although new methods of solving differential-algebraic equations (Gear, 1971c; Petzold, 1983) are capable of solving both types of problems. In dynamic simulation, the equations are usually integrated over a small time interval, h, called the integration steplength. This process is repeated starting from an initial time to a final time of integration. This provides a profile of the variables as a function of time. The value of the steplength depends on several factors, including the stiffness of the problem, the error criteria specified by the user, and the stability of the numerical algorithm used. These factors are discussed in Chapter 3.

In sections 2.2, 2.3 and 2.4, the solution methods mentioned above are described, first for flowsheeting systems and then applied to dynamic simulation systems. In each case, the methods of formulating the model equations for solution are described and their relative advantages and disadvantages discussed. Section 2.5 discusses the relative advantages and disadvantages of the various methods and the future trends.

2.2 THE SEQUENTIAL MODULAR APPROACH

This architecture has been used in the design of most flowsheeting systems. Examples include PROCESS (Brannock et al, 1979), FLOWTRAN (Proctor, 1983), FLOWPACK II (Berger and Perris, 1979). The main features of this approach are that each unit operation in a chemical plant, e.g. valve, reactor, distillation column, is modelled by systems of algebraic equations (eq. 2.1 with $\mathbf{y}'$ equated to zero). Several assumptions are often made which allow for easier and more efficient solutions. These include:

1. The model parameters associated with the units should always be defined and

should not be treated as unknowns. The same assumptions is made for variables associated with feed streams entering the plant from outside.

2. The information flow in the mathematical model coincides with the material flow in the chemical plant.

With these assumptions, computer subroutines are developed for these units and the equations are structured to calculate the output stream variables given the input stream variables and model parameters of the unit. The executive routine then calls each module in a predetermined sequence to solve the equations of the unit. The algebraic equations are solved for the algebraic variables within the unit. Recycle streams are torn and converged by an iterative process, using initial estimates provided by the user or defaulted by the system. Specifications, i.e. design constraints, must be solved iteratively as well. Details of this approach are provided in the review articles by Motard et al. (1975), Hlavacek (1977) and Rosen (1980). A book on process flowsheeting written by Westerberg et al. (1979) provides an extensive coverage.

2.2.1 APPLICATION TO DYNAMIC SIMULATION

Early process dynamic simulators have been designed using this approach (e.g. DYFLO (Franks, 1972); DYNSYS (Barney, 1975) and DYNSYL (Patterson and Rozsa, 1980)). The model equations (equations. 2.1a and 2.1b) are structured in modules. The executive routine calls the modules in a predetermined sequence with the values of the input variables at the current time t+h, and the values of the output variables at the previous time t, to compute the derivatives of the output variables. This is called derivative evaluation. The algebraic variables are computed in much the same way as in flowsheeting systems. If the flowsheet of the plant contains recycles, they are torn at time t+h and iteration proceeds around the recycle loops until successive values of the torn variable converge. This procedure is usually faster than the case of flowsheeting

systems because the recycle variable values at time t provide a better estimate at time t+h.

2.2.2 FORMULATION OF THE MODEL EQUATIONS

The differential equations in the model must be written in standard form as in equation 2.1a. In this case the system of differential equations can be written in the following form:

$$\mathbf{Ey'} = \mathbf{f(y,z,p,x},t) \qquad ..2.2$$

where **E** is the identity matrix.

However, most dynamic models results in differential equations not in standard form (e.g. models of seperation processes, see Holland and Liapis, 1983). They must therefore be manipulated upon (usually by approximations) to reduce then to standard form. The algebraic variables must be solved independently before derivative evaluation. For each model routine containing algebraic equations an iteration loop must be set up and solved using a convergence algorithm like Newton's or the quasi-Newton's method (Sargent, 1981).

2.2.3 COUPLED AND UNCOUPLED SOLUTION METHODS

The degree of coupling allowed between modules and between equations in a module during the solution process can affect the accuracy of the result as well as the choice of the numerical method of solution. In the uncoupled mode of solution, each module is simulated independently of the other modules starting from the initial time, t_{start} to the final time t_{end}. The danger in this mode of solution is that the modules are not really uncoupled and outdated values of the input variables are used in the calculation of the values of the output variables.

In the coupled mode of solution, the ouput variables of a module are calculated independently of the other modules at every step of the integration. Coupling is only achieved at the end of the step. The main advantages of the coupled and uncoupled modes of solution are that the maximum size of the problem to be handled depends on the size of the largest module and also, it is possible to use different integration algorithms for different modules depending on the nature of the problems (Hillestad, 1986). In DYFLO (Franks, 1972), explicit integration algorithms (explicit Euler and fourth order Runge-Kutta methods) are used in coupled mode of solution, although the new version, DYFLO2 (Franks, 1982) uses implicit Euler for stiff problems. DYNSYL (Patterson and Rozsa, 1980) provides for coupled, uncoupled and mixed modes of solution. In the latter case, the uncoupled mode is used for a sub-interval of the total time of simulation, with coupling achieved only at the end of the sub-interval. Cameron (1981) has investigated this type of approach in integration and concluded that they can produce inaccurate results. This is primarily due to the fact that during the sub-interval when the uncoupled mode is used, outdated values of the input variables are used in the calculations.

2.2.4 LIMITATIONS OF THE SEQUENTIAL MODULAR METHOD

Dynamic simulators designed using the sequential modular architecture inherit all the disadvantages of their flowsheeting systems counterparts. These include inefficiency in handling design and optimization problems and high order recycle flowsheet calculations as discussed by Westerberg et al (1979), Rosen (1980), Kuru (1981) and Cameron (1981). However, for dynamic simulation purposes one of their main limitations is that the unidirectional information flow requirement precludes the convenient modelling of flow-pressure interactions and complex flow networks, which could give rise to bidirectional flow (Figure 2.2) (Aylott et al, 1985). The basic problem in flow pressure modelling is that flows are dependent upon downstream pressures. Where these are not fixed they must be determined by passing them back to a flow calculator, thus creating

an information recycle, in spite of the fact that there are no process recycles. The calculation of flowrate, F1 in Figure 2.2 requires the value of pressure, P2 of the next unit in the sequence to be processed as well as the upstream pressure, P1.

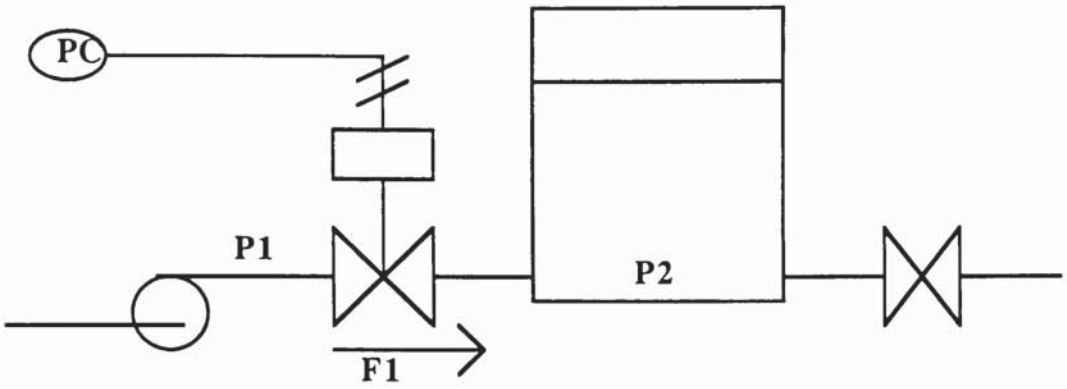


Fig. 2.2 Flowsheet of a plant illustrating bidirection flow

Flow-pressure difficulties can be avoided by neglecting flow changes compared to composition or holdup changes. In many applications, this type of approximation may not be acceptable, in which case this approach cannot be used.

2.3 THE EQUATION-ORIENTED APPROACH

In equation-oriented simulation, all the equations modelling the chemical process are solved simultaneously. This means that a large system of nonlinear algebraic equations containing from hundreds to thousands of variables has to be solved at every iteration. Two approaches for solving these equations can be distinguished. The first is called tearing (e.g. SPEED-UP (Perkins and Sargent, 1982)). Here values are guessed or torn, for a number of variables sufficient to permit values of the remaining variables to be found by solving a sequence of small, usually one-variable problems. The tearing approach involves development of an appropriate solution strategy and an information flow pattern for the particular problem. This involves deciding which variables to tear,

which equations to solve for which variables (output set), and which sequence to solve them (precedence ordering). What is needed is a systematic and efficient procedure for finding a solution strategy that will converge reliably and rapidly to the solution, but this problem has not been fully solved (Stadtherr and Hilton, 1982 ; Westerberg et al, 1979).

The other approach is the use of the Newton-Raphson, quasi-Newton or variants of these methods for the simultaneous linearization of the equations and then iteration on all the variables This is generally called the quasi-linear method. It means the solution of a huge set of sparse linear algebraic equations at every iteration. Applications of the Newton-Raphson method to solve some specific problems have been summarised by Westerberg et al. (1979). The computational problem with the quasi-linear methods is to find a strategy to solve the large sparse linear systems that arise. A good initial guess is needed for the straightforward Newton-Raphson method. More reliable methods have been suggested by a number of workers (Gorczinski and Hutchison, 1978 ; Westerberg and Director, 1978 ; Chen and Stadtherr, 1981). All these methods hold much promise for flowsheeting systems.

2.3.1 PROBLEMS WITH THE EQUATION-ORIENTED APPROACH

Although equation-oriented simulation has potential to overcome all the problems associated with the sequential-modular approach, its acceptance as a commercially viable alternative has been slow due to a number of serious disadvantages associated with it. Notable among them are:

1. The executive programs of this method are very complex, especially the setting up of the equations to be solved from the modules.

2. They are more difficult to debug.

3. The storage requirements are very excessive, making it difficult to simulate

realistic process models.

4. The solution methods and algorithms are very complicated and solution is not guaranteed because general-purpose software is used.

5. If the solution fails the user is left with little or no useful information.

The reviews by Shacham et al. (1982), Perkins (1984), and Westerberg and Benjamin (1985) which discuss the main developments in equation-oriented flowsheeting systems conclude that recent developments in computer technology and numerical solution algorithms have almost eliminated or drastically reduced the significance of these drawbacks. This has been confirmed by the number of process flowsheeting systems being developed. These include SPEED-UP (Perkins and Sargent, 1982) and TISFLO II (Van Menlebrouk et al, 1982), which are industrial simulators. Others are ASCEND II (Kuru, 1981), QUASILIN (Field et al, 1985 and Hutchison et al, 1986), FLOWSIM (Shacham et al, 1982) and SEQUEL (Stadtherr and Hilton, 1982) which are at various stages of development. Some of the new developments can be summarised as follows:

1. Automatic initialization routines that find initial guesses which are good enough for quasi-linear methods to converge rapidly (Locke, 1981)

2. Availability of cheap and powerful microcomputers with large storage capabilities, providing simulation engineers with more access to computers than ever before.

3. The development of sparse matrix packages specifically for flowsheeting systems (Stadtherr and Wood, 1984), which can be used to reduce the huge storage requirements demanded by the use of equation-oriented methods.

4. The organization of the model equations of unit operations into modules. This relieves the user of the task of writing his own model every time he uses the simulator. It also presents a modular appearance to the user

5. Development of more reliable equation solvers (Sargent, 1981; Shacham et al. 1982 and Paloshi, 1982), which can handle the large, sparse system of nonlinear algebraic equations that arise.

It can be seen that equation-oriented simulation is now a commercially viable method. What is needed is implementation and efficient incorporation of these new ideas into a working equation-oriented package.

2.3.2 APPLICATION TO DYNAMIC SIMULATION

The application of the equation-oriented approach to dynamic simulation amounts to solving the differential-algebraic equations that model the dynamic behaviour of chemical processes. The solution of differential-algebraic systems can be achieved in a variety of ways. In most cases the formulation used is determined by the available solution algorithm.

2.3.3 MODEL FORMULATION STRATEGIES

The general DAE problem can be formulated for solution using different strategies. For the equation-oriented solution method, three types of formulations can be distinquished;

1. Formulation I

All the differential equations are solved simultaneously while the coupled algebraic equations are solved internally within an inner loop of the iteration process. This requires the differential equations to be in standard form. Reliable codes for solving

ODEs written in standard form are available (e.g. EPISODE (Byrne and Hindmarsh, 1975)). Dynamic simulators designed using this formulation include ACES (Kocak, 1980), FLOWSIM V4.0 (Babcock, 1981) and all CSSL-type dynamic simulators (e.g. ACSL (Mitchell, 1978). This can be regarded as the "outer-inner" convergence loop approach, in which all the algebraic equations are partially decoupled from the differential equations and solved within the inner loop while the differential equations must be written as in equation. 2.3

$$\mathbf{E}\mathbf{y}' = \mathbf{f}(\mathbf{y},\mathbf{x},\mathbf{p},\mathbf{z},t) \qquad ..2.3$$

where

E is an identity matrix.

Another drawback of this method is that all the algebraic variables become local variables and are thus not "available" for global manipulation as may be required in design and optimization problems. Handling of design and optimization problems require the use of "controlled simulation" as in sequential-modular methods. This has been described in detail by Hlavacek (1977), Westerberg et al. (1979) and Cameron (1981). In ASCEND II (Kuru, 1981) and DYNAMIC QUASILIN (Smith and Morton, 1987) a new variable, known as a velocity variable, is introduced for every differential equation. Equations 2.1a and 2.1b become

$$\mathbf{v} = \mathbf{y}' \qquad ..2.3a$$

$$\mathbf{v} - \mathbf{f}(\mathbf{y},\mathbf{z},\mathbf{p},\mathbf{x},t) = \mathbf{0} \qquad ..2.3b$$

$$\mathbf{g}(\mathbf{y},\mathbf{z},\mathbf{p},\mathbf{x},t) = \mathbf{0} \qquad ..2.3c$$

This increases the dimension of the problem by introducing an extra variable and an extra equation for each derivative term. Also, equations which contain more than one derivative term must be manipulated on in order to transform them to state variable form.

2. Formulation II

The DAEs are assumed to be in linearly implicit form

$$\mathbf{Dy'} = \mathbf{f(y,p,x,z,t)} \qquad ..2.4$$

where

D is a matrix whose element, d_{ij} is either 0 or 1.

Most codes for solving system (2.4) impose the restrictions that the matrix **D** be nonsingular (e.g. LSODI (Hindmarsh, 1983)). It may be possible to solve system (2.4) for the case where **D** is singular, provided that all the derivatives be provided. Even in this case solution is not guaranteed (Petzold, 1983). Using this formulation means that models of separation processes where the coefficients of the derivatives can be variable are not solvable although they can be manipulated upon to bring them to the form (2.4). Examples of dynamic simulators using this approach include BOSS (Joglekar and Reklaitis, 1984).

3. Formulation III

This formulation imposes no restrictions on the model. The equations can have more than one derivative term in each equation. The derivatives can have variable coefficients as well. The general form of these equations are:

$$\mathbf{Au'} = \mathbf{f(u,p,x,t)} \qquad ..2.5$$

where

u a vector of all the state variables, both differential and algebraic,

u' vector of the derivatives of the state varibles. $\mathbf{u}_i$ is an algebraic variable if all $\mathbf{u'}_i$ is zero in all the equations, otherwise it is differential,

A matrix of coefficients of the derivatives, which may be constants, variable or form a singular matrix.

This formulation has several advantages over the others:

1. There is no need to manipulate the derivative terms to bring them to state variable form as in formulation I.

2. The matrix **A** in equation (2.5) can be singular.

3. This formulation provides a consistent way of writing the model eqations in the modules. All the equations can be written in the form

$$\mathbf{F(u',u,p,x,}t\mathbf{)} = \mathbf{0} \qquad ..2.6$$

That is, no distinction is made between the algebraic and differential equations as in formulations I and II.

4. The same model equations can be used for both steady state and dynamic simulations by equating all derivatives terms to zero before function or Jacobian matrix evaluations. This formulation does not require equations to be in state variable form and no new variables or equations are introduced. This is the formulation used in DASP, and we shall see later in Chapter 6 that it offers advantages in the structuring of the equations to be solved within the modules.

5. Design constraints can be added to the system of equations as

$$\mathbf{H(w)} = \mathbf{0} \qquad ...2.7$$

where

w is a vector of constraint variables,

H the constraint equations.

6. Optimization can be performed by adding the objective function and inequality constraints to the model equations which form the bulk of the equality constriants as in

$$\text{Max} \quad c(y)$$

s.t.

$$g(y) = 0 \qquad \ldots 2.8$$
$$h(y) \geq 0$$

and then a suitable optimization algorithm (Westerberg, 1980) can be used to solve these equations.

2.3.4 MODULARLY ORGANIZED E-O APPROACH

One of the main criticisms of the equation-oriented approach is its apparent "unfriendliness" to the user, especially the need to supply all the model equations for every simulation problem. This can be overcome by providing models of individual unit operations in modules, which either calculate the derivatives of the state variables or the residuals of the equations. However, organization of the equations in modules presents several problems. For example, the assembling of the model equations for a particular problem from the relevant modules and the transfer of variables and parameters between modules become very difficult.

Several strategies have been devised to handle this problem for the different simulators. In ASCEND II (Locke, 1981), a combination of "equation generators, equation and variable packets" have been used. A generator is defined as a collection of Fortran subroutines that perform specific tasks such as function evaluation or evaluation of terms for the Jacobian matrix for the specific equation, with the help of equation and variable packets. The complexity and large number of these generators give an indication as to the dimension of the problems of setting up the equations to be solved. However, if

formulation III described above is used, generation of the equations becomes relatively easier as all that is required in all cases is the calculations of the residuals of the equations.

2.4 SIMULTANEOUS MODULAR METHOD

This architecture requires that all the equations modelling the process plant be solved simultaneously, as in the equation-oriented approach. However, the types of equations and their formulations are quite different. Here two types of models are used for a process unit, a simple and a rigorous model. The rigorous models consist of the describing equations along with algorithms designed to solve them. These rigorous models are the modules of the sequential modular approach. The models are used to generate parameters of the corresponding simple models which relate the outputs to the inputs as for example in the linear case

$$\mathbf{Y} = \mathbf{AX} \qquad \ldots 2.10$$

where

Y the output variables,

X the input variables,

A the coefficients of the simple model which relate the outputs to the inputs.

There are thus two levels of computation, a module level in which the coefficients of the simple models are generated and a flowsheet level where the simple equations are solved simultaneously for the output variables (Chen and Stadtherr, 1985).

2.4.1 APPROACHES TO SIMULTANEOUS MODULAR METHOD

The various techniques proposed for the simultaneous modular approach differ in the way the coefficients required for the simple models are generated, the numerical methods used to solve the simple models and whether all the connecting streams are iterated on or

only an appropriate set of tear streams. The simple models may take the form of linear models (Mahalec et al, 1979 ; Sood et al, 1979) or they can be approximate engineering models in which nonlinearities are represented more accurately. They use smooth functions with analytical partial derivatives and avoid time consuming thermodynamic physical property calculation routines that use non-smooth functions (e.g. ASPEN (Evans et al, 1979)).

Chen and Stadtherr (1985) discussed the various techniques used to generate the coefficients of the linear equations. Recent studies by Mahalec et al (1979), Perkins et al (1979) show that the simultaneous modular approach has a promising future and provides a better alternative to the sequential-modular method. The simultaneous modular approach is an attempt to combine the good features of the modular and equation-oriented approaches. For example, there is no need for the costly control loops used in sequential-modular approach to handle design and optimization calculations, since design specifications can be handled directly at the flowsheet level. Also, converging the linear equations is relatively easier than converging the nonlinear equations of the equation-oriented simulation. Finally, there is a saving to be made by using the existing modules of the sequential modular simulators.

2.5 DISCUSSION

As explained in Section 2.2, sequential-modular simulation derives its attractiveness from the modular organization of the model equations and the unit-by-unit solution approach which makes it possible to use reliable solution algorithms where desirable. Also, the modules can be tested individually. If the solution fails, it is easier to diagnose the failure. Moreover, large investment has already been put into the development of such programs and these have proven industrial use. There are, however, the problems with this approach which have already been mentioned. A lot of progress has been made

to overcome some of these problems (Clark and Reklaitis, 1984), but this has not gone far enough to outweigh the advantages of using the simultaneous methods. It is a difficult task to modify a sequential-modular steady state simulator to perform dynamic simulation, which will require the writing of new dynamic model routines and new interface programs and solution algorithms which are different from their steady state counterparts. Two attempts have been made (Bobrow et al, 1971 and Alyott et al. 1985). Both earlier forecasts (Motard et al, 1975 ; Hlavacek, 1977) and recent reviews (Locke, 1981 ; Shacham et al, 1982 ; Westerberg and Benjamin, 1985) all agree that the future of simulation is to move towards the use of the simultaneous methods.

Within the equation-oriented approach, the quasi-linear methods seem to have an edge over the tearing rival. Tearing needs an efficient tearing algorithm, and a new solution strategy that converges rapidly and these problems have not yet been fully solved (Chen and Stadtherr, 1985). Also, it may be necessary to have several versions of the same model for different problems, which will create many redundant equations to overwhelm the computer memory. This will then undermine the advantage that tearing does not require the use of the Jacobian matrix which takes large computer memory. The Newton-Raphson method converges very rapidly if the initial values are very close to the solution. Recently, attempts have been made to find close-enough initial guesses for this method (Locke, 1981). Moreover, several new hybrid methods have been developed which claim high rate of convergence from poor initial guesses (Westerberg and Director, 1978 ; Chen and Stadtherr 1981).

In spite of the relative advantage of the simultaneous modular method over the equation-oriented approach because of the use of existing modules of sequential modular simulators, there are some disadvantages associated with it. Firstly, the simple models are written in terms of material and enthalpy flow. On the other hand, temperature does not appear in the simple models, and hence is not "available" for design and optimization calculations, except by the indirect method of using control loops. Secondly, writing the

simple models can be difficult. If not written correctly the system of equations will not converge (Westerberg et al, 1979). Therefore, for dynamic simulation purposes, the equation-oriented approach holds more promise than either of the other two methods. This is the strategy used in DASP, and is presented in this thesis.

CHAPTER 3

NUMERICAL SOLUTION METHODS

3.1 INTRODUCTION

Process dynamic simulation involves solving the dynamic model equations of the chemical process. These equations can vary between ordinary differential, partial differential, integral and algebraic equations depending on the process modelled and upon the level of detail chosen for the model. We shall consider model equations to be either first order ordinary differential equations (ODEs) or a combination of first order ODEs coupled with non-linear algebraic equations, known as differential-algebraic equations (DAEs). This is because all other deterministic forms of model equations can be converted or transformed to ODEs or DAEs (Carvar, 1981 ; Carnahan and Wilkes, 1980). Examples of this transformation are given in Appendix B1. Also, standard software for solving ODEs is available and reliable and software for solving DAEs is now becoming available. We are only concerned with initial value problems of the form

$$\mathbf{F}\ (\mathbf{y}',\ \mathbf{y},\ t) = \mathbf{0} \qquad .. \ 3.1$$

given

$$\mathbf{y_0} = \mathbf{y}\ (t_0)$$

where

$\mathbf{y}$ is a vector of state variables (both differential and algebraic) of order N

$\mathbf{y}'$ is the vector of the derivatives of the state variables of order N

t is the time

$\mathbf{F}$ are the state equations.

In equation 3.1, y_i is an algebraic variable if no terms in y'_i are non-zero and F_j is an

algebraic equation if no terms in $\mathbf{y'}_i$, i = 1, N appear in the equation.

The numerical solution of these equations lie at the core of any dynamic simulator. In Section 3.2, a brief description of the Backward Differentiation Formula, BDF (Gear, 1971a) for solving differential equations is given. It is shown how BDF can be extended to solve DAEs using the idea first introduced by Gear (1971c). Section 3.3 analyses the solvability of DAEs in the light of recent investigations by several workers (Petzold, 1982 ; Pantelides et al, 1987 ; Gear and Petzold, 1984) and presents some of their findings.

Dynamic simulation also involves the handling of events, and Section 3.4 presents the mathematical treatment of state events, which are analogous to integration over discontinuities. A suitable algorithm is sought for implementing in the event processing package, described in Chapter 8. One of the advantages of the implicit formulation of model equations as described in Chapter 2 is the ability to use the same model equations for both steady state and dynamic simulations without any structural modifications. Steady state simulation requires the solution of non-linear algebraic equations and the numerical methods for handling them are briefly described in Section 3.5. This chapter aims to find the necessary numerical methods and algorithms for implementing in the DASP package.

3.2 THE NUMERICAL SOLUTION OF ODES

The initial value (IV) ordinary differential equation

$$\mathbf{y'} = \mathbf{f}\ (\mathbf{y},\ t) \qquad .. \ 3.2$$

given $\mathbf{y_0} = \mathbf{y}\ (t_0)$

can be solved by stepping algorithms that start with the initial condition, y_0 and generate approximations, $\mathbf{y}_i$ of the dependent variable $\mathbf{y}(t_i)$ at discrete values of the independent

variable, t_i, i = 0, M with

$$t_{i+1} = t_i + h_i \qquad .. \ 3.3$$

where h_i is the stepsize for the i^{th} step. The numerical algorithms and methods for solving equation 3.2 are described in most numerical analysis text books, (see Carnahan and Wilkes (1980)). If all the h_i are equal to h, then $t_i = t_o + ih$, i=0, M. The error generated by the application of these algorithms, known as the global truncation or discretization error can be represented by

$$e_i = y_i - y(t_i) \qquad .. \ 3.4$$

This has two main components, namely the propagated error which results from approximations made in previous steps and a local error, which is the error that would be observed if all the previously determined variable values were error-free. The local error has two components, namely the truncation error due to the numerical approximation method used and the rounding error due to only carrying a finite number of significant figures.

One of the simplest numerical methods for solving the IV problem is the Euler method, which has the form

$$y_n = y_{n-1} + h_n f(y_{n-1}, t_{n-1}) \qquad .. \ 3.5$$

and defines a recurrence relation. In practice Euler's method is seldom used, since in order to obtain the necessary accuracy, small steps, h_n must be taken. This results in unreasonable computing time and also loss of precision due to numerical roundoff. Instead, higher order formulas like the Runge-Kutta (Carnahan & Wilkes, 1980), or Adams methods (Shampine & Gordon, 1975) are used. Generally, numerical methods

for solving the IV problem can be classified as either explicit or implicit. In explicit methods, the formula written in the form of equation 3.5 does not have y_n, which is the value to be calculated, in the right hand side of the equation. On the other hand, in implicit methods, y_n appears on the right hand side of the equation as well. This requires iterative solution.

3.2.1 STIFFNESS OF ORDINARY DIFFERENTIAL EQUATIONS

One of the characteristics of ODEs which make them difficult to solve is the problem of stiffness. Mathematical models of many chemical systems such as reactors, separation processes, control systems exhibit this property. Stiffness is the presence of time constants of different orders of magnitude. For example, in the kinetics of large multicomponent reactions, it is very likely that some of the reaction rate constants are substantially larger than others (Farrow and Edelsen, 1974). Also the models of distillation columns exhibit this property, especially with high product purity or low relative volatilities. In general, systems with widely varying time constants have some of their variables decaying faster than others. Mathematically, we can think of stiffness as a property in which the local Jacobian matrix of the system of equations

$$J_{ij} = \frac{\partial f_j}{\partial y_i}, \qquad i,\ j = 1, N \qquad \text{.. 3.6}$$

contains eigenvalues which are substantially different in magnitude. A measure of stiffness is the stiffness ratio, SR, defined as

$$SR = \max_{i=1,N} (\mathrm{Re}\ (\lambda_i)) / \min_{i=1,N} (\mathrm{Re}\ (\lambda_i)) \qquad \text{.... 3.7}$$

where

λ_i is the i^{th} component eigenvalue of the Jacobian matrix.

Stiff problems are difficult to solve because integration algorithms must use small stepsizes because of the rapidly decaying terms while at the same time integrate over a long period to handle the slowly decaying terms. Numerical methods behave differently with regard to stiffness. Explicit methods are constrained in the choice of the maximum stepsize to use according to the relation (Gear, 1971a)

$$| h (\lambda_{max}) | < C \qquad \ldots \ 3.8$$

where C is a small number between 1 and 10. Higher stepsizes cause the algorithm to be unstable. In Euler's method, for example, C = 2. One of the methods of overcoming the problem posed by stiffness is to eliminate the faster growing components by steady state approximation techniques. This method, though it works well for some specific problems, is not accurate and reliable enough (Edsberg, 1981).

Stiff problems are today tackled using implicit methods which have no limiting constraints on the maximum stepsize to use. Of the many implicit methods capable of handling these problems, the most popular are the semi-implicit Runge-Kutta methods (Michelsen, 1976 ; Bui, 1981 ; Chan et al, 1978), diagonally implicit Runge-Kutta methods (Alexander, 1977 ; Cameron, 1981) and the backward differentiation formulas (BDF) first implemented by Gear (1971b). Runge-Kutta methods are less successful than BDF methods, as they require an accurate Jacobian matrix and the differential equations to be in state variable form (Feng et al, 1984).

3.2.2 A BRIEF DESCRIPTION OF THE BDF METHODS

The backward differentiation methods (BDF) are a family of algorithms which can be derived from the linear multistep formula (Gear 1971a):

$$\mathbf{y}_n = \sum_{i=1}^{q} a_i \mathbf{y}_{n-i} + h \sum_{i=o}^{q} \beta_i \mathbf{f}_{n-i} \qquad \ldots 3.9$$

where

$\mathbf{f}_j = \mathbf{f}(\mathbf{y}_j, t_j)$

q the order of the method.

The coefficients $\{a_i\}$ and $\{\beta_i\}$ are 2q + 3 parameters to be determined such that if the solution $\mathbf{y}(t)$ of an initial value problem is given by a polynomial of degree k, then equation 3.9 gives the exact solution. A number of BDF formulas can be derived by making suitable choices of $\{a_i\}$ and $\{\beta_i\}$. For example, Gear's k^{th} order corrector algorithm (Gear, 1971a) is an implicit algorithm which is obtained by setting

$q = k, \beta_1 = \beta_2 = \ldots = \beta_k = 0$

resulting in the formula

$$\mathbf{y}_n = \sum_{i=1}^{k} a_i \mathbf{y}_{n-i} + h \beta_o \mathbf{y}'_n \qquad . 3.10$$

Equation 3.10 is an implicit formula which requires iteration, but in practice this equation is combined with the equivalent explicit formula in a predictor-corrector iteration scheme, where the explicit form, or predictor, is used to provide a good initial guess of $\mathbf{y}(t)$ at time t_n for the corrector equation. If we rearrange equation 3.10 as

$$\mathbf{y}'_n = \frac{1}{h\beta_o} \left[\mathbf{y}_n - \sum_{i=1}^{k} a_i \mathbf{y}_{n-i} \right] \qquad . 3.11$$

and substitute it into the equation

$$F\ (y',\ y,\ t) = 0 \qquad .3.12$$

then we obtain a system of algebraic equation

$$F\left[\left(\frac{1}{h\beta_o}\right)\left(y_n - \sum_{i=1}^{k} a_i\, y_{n-i}\right),\ y_n,\ t_n\right] = 0 \qquad .. \ 3.13$$

If y_n is an approximation appearing in the m^{th} iteration of a Newton's method solution of equation 3.13, then

$$\left[\frac{\partial F}{\partial y'} + h\beta_o \frac{\partial F}{\partial y}\right]_n^m \left(y_n^{(m+1)} - y_n^{(m)}\right) = -\left[h\beta_o\, F\right]_n^m \qquad .. \ 3.14$$

The matrix superscripts (m), indicate evaluation using the m^{th} estimates of y_n, y'_n and t_n. In practice, however, it is usually sufficient to use the Jacobian matrix for several iterations without re-evaluation (see Gear, 1971a; Byrne and Hindmarsh, 1975). This significantly reduces the work involved in solving equation 3.14.

Two techniques are commonly used to implement variable-step variable-order BDF methods. One technique is based on a fixed coefficient interpolation formula (i.e. $\{a_i\}$ and $\{\beta_i\}$ constant in equation 3.10) and uses interpolation to generate approximations to the solution at evenly spaced past points. Examples of this implementation are the DIFSUB (Gear, 1971b) and the GEAR (Byrne et al, 1977) codes. The other technique uses variable coefficient interpolation formula (i.e. $\{a_i\}$ and $\{\beta_i\}$ calculated at every

step). The codes EPISODE (Byrne and Hindmarsh, 1975) and BDF (Brayton et al, 1972) are two examples of this implementation. It has been proved by several workers (Brayton et al, 1972 ; Gear and Tu, 1974 ; and Byrne and Hindmarsh, 1975) that the variable coefficient implementation is more stable than the fixed coefficient formulation, a property very important for solving stiff problems.

It is also possible to implement BDF using a fixed leading coefficient formula. Equation 3.10 can be rewritten as:

$$\alpha_{n0}\, y_n = \sum_{i=1}^{k} a'_i\, y_{n-i} + h y'_n \qquad .. \quad 3.15$$

where

$$\alpha_{n0} = \frac{1}{\beta_0}, \quad a'_i = \frac{a_i}{\beta_0}, \quad i = 1, k$$

Here α_{n0} is called the leading coefficient. In a fixed coefficient implementation, α_{n0} and the a_i's are changed only when the order, k, and thus the formula itself, is changed. On the other hand, the fixed leading coefficient implementation calculates all but the leading coefficient, α_{n0} at every step. Jackson and Sacks-Davies (1980) claim that this formula tends to be more stable than the fixed coefficient implementation and in some respects is more efficient than the variable coefficient formula used in codes like EPISODE (Byrne and Hindmash, 1975).

The representation of the approximating polynomial can take various forms. In EPISODE and DIFSUB, the polynomial is represented by an array of the variables and its scaled derivatives

$$Z_{n-1} = [y_{n-1},\ h_n\, y^{(1)}_{n-1},\ h_n{}^2\, y^{(2)}_{n-1},\ h_n{}^k\, y^{(k)}_{n-1}] \qquad 3.16$$

where $y_{n-1}^{(i)}$ is the i^{th} derivative of the polynomial evaluated at t_{n-1}. Some codes eg. DASSL (Petzold, 1983) use a divided difference representation

$$[y_n, y_{n-1}, y_{n-2}, \cdots, y_{n-k-1}] \qquad .. \ 3.17$$

or its scaled form. The error estimate generally used is based on the modified Milne estimate (Gear, 1971a). In this estimate, the error is based on the difference between the predicted and corrected values of the variables, which can be represented as:

$$\mathbf{Est}_n(k) = C(k).(y_n - y_{n,0}) \qquad .. \ 3.18$$

where:

$\mathbf{Est}_n(k)$ is a vector of the estimates of the truncation error for step n with order k.

y_n is the vector of the corrected values

$y_{n,0}$ is the vector of the predicted values

C(k) is the error coefficient.

Detailed implementation of the BDF methods can be found in the codes referenced.

3.3 ANALYSIS OF APPROACHES FOR SOLVING DAES

Earlier designs of dynamic simulators used different methods to solve the algebraic and differential equations. That is, ODE solvers were used to tackle the differential equations in the model equations while the algebraic equations were solved using nonlinear equation solvers. As discussed in Chapter 2, this is not only inconvenient but may result in loss of accuracy. It also makes it difficult to use the same model equations for both steady state and dynamic simulations. Several researchers have investigated the extension of ODE solvers for the simultaneous solution of differential-algebraic systems.

Holland and Liapis (1983) and Cameron (1981) presented the DAE versions of the semi-implicit and diagonally implicit Runge-Kutta methods respectively. For reasons mentioned in Section 3.2.1, Runge-Kutta methods are not competitive with BDF methods. Codes based on the BDF method have been described by Brown and Gear (1973), Starner (1976), Curtis (1978), Soderland (1980), Hindmarsh (1981) and Petzold (1983). They are all based on the technique first introduced by Gear (1971c).

The central idea is that the derivative, **y'** can be approximated by a linear combination of the solution **y**(t) at the current and several previous mesh points, resulting in a non-linear algebraic equation which can be solved by Newton's methods or variant thereof. For example, replacing the derivative in equation 3.1 by the backward difference at time t_n we obtain the first order formula:

$$\mathbf{F}\left(\frac{y_n - y_{n-1}}{h_n}, y_n, t_n\right) = \mathbf{0} \qquad . \ 3.19$$

which can be solved using Newton's method as:

$$y_n^{(m+1)} = y_n^{(m)} - \mathbf{G}^{-1}\,\mathbf{F}\left[\frac{y_n^{(m)} - y_{n-1}}{h_n}, y_n^{m}, t_n\right] \qquad . \ 3.20$$

where

$$\mathbf{G} = \left[\frac{\partial \mathbf{F}}{\partial \mathbf{y}'} + \frac{1}{h_n}\frac{\partial \mathbf{F}}{\partial \mathbf{y}}\right] \qquad .3.20a$$

m is the iteration counter.

In practice, the approximation is done using higher order BDF formula because the first order backward difference is not accurate enough.

3.3.1 SOLVABILITY OF DAES USING BDF METHODS

The solvability of differential-algebraic systems using ODE methods have been investigated by Gear and Petzold (1984), Petzold (1982), Campbell and Petzold (1982), Reinboldt (1984) and Sincovec et al, (1981). The aims were to find reasons why DAE systems are difficult to solve and why many ODE solvers adapted to solve DAEs fail. It is interesting to note that very little is known of the causes of these failures and the findings are not conclusive. However, all agree that certain classes of DAEs are not solvable by available codes. If we represent DAEs as

$$\mathbf{E}\ \mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t), t) \qquad .\ 3.21$$

where

$\mathbf{y}(t)$ is a N-dimensional vector of state variables

$\mathbf{E}$ is a N x N matrix of coefficients.

If $\mathbf{E}$ is constant and of the form

$$\mathbf{E} = \begin{array}{c|c} \mathbf{I}^{(s)} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array} \qquad .. \ 3.22$$

where $\mathbf{I}$ is the identity matrix of order s ($1 \leq s < N$) then the system is solvable (Gear and Petzold, 1984). In general, let the function $\mathbf{F}(\mathbf{y}(t),t)$ be continuously differentiable with respect to $\mathbf{y}(t)$. If we consider the Jacobian matrix of the system which is partitioned like $\mathbf{E}$ (equation 3.22).

$$\mathbf{J}(t) = \begin{array}{c|c} \mathbf{J}_{11}(t) & \mathbf{J}_{12}(t) \\ \hline \mathbf{J}_{21}(t) & \mathbf{J}_{22}(t) \end{array} \qquad .. \ 3.23$$

where $\mathbf{J}_{11}(t)$ is an sXs matrix, then equation 3.21 is considered solvable (Petzold, 1982; Caracotsios and Stewart, 1985) if

$$\det \mathbf{J}_{22}(t) \neq \mathbf{0} \text{ for all } t \qquad .. \ 3.24$$

Under this condition, the solution obtained by a BDF algorithm of order k, with $k < 7$ and fixed step size, h, converges to $O(h^k)$, if all initial values are correct.

Differential-algebraic equations are not ODEs and most solution algorithms available presently are not very reliable. The problem is made easier by the fact that a large number of the systems encountered in chemical engineering analysis either fall into the class of solvable systems or can be transformed into such systems (Gallun and Holland, 1982; Holland and Liapis, 1983; Gear and Petzold, 1984). This class of solvable systems are termed index 1 systems (Petzold, 1982). Pantelides et al (1987) have looked into the problem of solvability of chemical engineering systems and their classification into indexes. They define the index of a DAE system as the minimum number of differentiations with respect to time that the system equations have to undergo to convert the system into a set of ODEs. Using this definition, ODEs are index zero systems and all dynamic simulation problems fall into the class of solvable index 1 systems. One class of chemical engineering problems, which they identified as potential higher index problems (index greater than one) is called the dynamic design problem, in which one or more input variables are unknown, and it is desired to determine them so that a given variation in one or more output variables is achieved.

Higher index problems are difficult to solve and in some cases are not solvable by available DAE solvers. In investigating this problem, Petzold and Gear (1984) suggest possible transformations of the original system of equations, which will convert them to lower index problems. This may involve differentiation of the algebraic equations or reduction of the system of equations by eliminating some of the algebraic variables. They admitted that these transformations are not always possible. Petzold (1982) analysed the problem with regard to the strategies and methods used in the design of the DAE solvers, especially the step size changing strategies and error control of the integrator, and suggested new approaches to this problem. Pantelides et al (1987) found a connection between the index of the problem and the difficulties in providing consistent initial conditions. They presented an algorithm for providing consistent initial values for

all the variables and derivatives for higher index problems. At present, these investigations are still going on and not much has been done to implement some of the findings and suggestions put forward. It is hoped that better codes will be designed to handle higher index problems.

Fortunately, dynamic simulation problems fall into the class of solvable problems or can be transformed to such problems. The real problem is in dynamic design problems. An analysis of the examples of dynamic design problems presented in Pantelides et al (1987) suggests that the modelling assumptions and the arbitrary choice of design variables, without regard to fundamental physical and chemical constraints, result in higher index problems. Firstly the assumption that there could be almost unlimited freedom in specifying and solving design problems is not the practice in chemical engineering design. Chemical Engineering design is partly an art, which is why the choice of which variables to regard as design variables are usually based on experience and engineering judgement. Although the equation-oriented approach offers almost total flexibility in specifying design variables, in practice there are both physical and chemical constraints which will inevitably limit this choice. Secondly, detailed modelling of systems can result in lower index problems, while certain assumptions help to create higher index systems. For example, neglecting the hydraulic relationships and vapor hold-up on the trays of separation processes may result in higher index systems as noted by Pantelides et al (1987), while these problems have been solved very easily by Gallun and Holland (1982) using available DAE solvers by modelling these systems in detail.

In summary, the results of these investigations are as follows:

1. Certain classes of DAE systems cannot be solved by variable stepsize BDF methods, which use certain stepsize changing strategies (see for example Jackson and Sacks-Davies, 1980) to accelerate the solution. For example, some step size selection algorithms assume that errors are O (h^{k+1}), where k is the order of the method, but in fact, the error has been found to be independent of the stepsize after the first step for

certain DAE systems (Petzold, 1982).

2. In trying to overcome some of the difficulties presented by DAE systems, new error control strategies have been suggested. For example, Sincovec et al (1981) and Brown and Gear (1973) do not control the error in the algebraic components. This was motivated by the observation that errors in algebraic components have a different asymptotic behaviour to those of differential components, so that error control strategies which assume the same behaviour would have difficulty controlling the errors in these components. This strategy is justified if the errors only affect the solution locally but are not propagated to the differential components. Also error control in the algebraic component can be neglected if we are not interested in the values of those components. This must be applied with care. DAEs can be thought of as limiting cases of stiff systems, in which the stiffness ratio has become infinite. It is possible that the algebraic components may behave differently in different regions along the interval of integration. In this case errors may be introduced which can then be propagated to the other components (Petzold, 1982).

3. Error estimates based on the difference between the predictor and corrector normally implemented in most BDF codes may be grossly inaccurate, and thus cause codes to fail unnecessarily. New error estimates need to be used for DAE solvers (Petzold, 1982). There has been no progress on this currently.

4. Most codes used in equation-oriented process dynamic simulators can only solve a restricted class of DAEs. For example, LSODI (Hindmarsh, 1981) is designed for linearly-implicit systems, for which the coefficient matrix of the derivative is not singular. In other codes, fixed coefficient formulas are used which can be unstable for certain stiff problems as discussed in Section 3.2.1.

However, DASSL (Petzold, 1983) can handle truely differential-algebraic systems of index 1. It uses a fixed leading coefficient BDF formula. It has been extensively tested

on a variety of problems by several users (Leis and Kramer, 1985; Caracotsios and Stewart, 1985). There are still problems which DASSL cannot tackle. Some of the causes of these failures have been identified by Rheinboldt (1984), who claims that his new code, PITCON (Rheinboldt and Burkardt, 1983) overcome most of the problems which make DASSL fail.

3.4 THE HANDLING OF DISCONTINUITIES

In dynamic simulation studies, the model equations frequently contain discontinuities in the form of switches which are thrown when certain conditions are fulfilled. For example, a safety valve can activate when the pressure reaches a given value, the threshold value, thus changing the system.

Consider the solution of the ordinary differential equation:

$$y' = f(y, t) \qquad . \ 3.25$$

$$\text{given } y(t_0) = y_0$$

using a discrete variable method. A unique solution is guaranteed if the function, f(y,t) (right hand side) is bounded and continuous and satisfies the Lipschitz condition:

$$\| f(y_1, t) - f(y_2, t) \| < L \| (y_1 - y_2) \| \qquad . \ 3.26$$

for some $L < \infty$. If f (y, t) is discontinuous at a point t = b for $a \leq b \leq c$, then clearly the Lipschitz condition is not fulfilled at t = b and b is called the switching point, defined by a point constraint. The point constraint can be given by an algebraic equation:

$$h(y(t_b), t_b) = 0 \qquad . \ 3.27$$

or an integral equation

$$\int_a^b y(s)\,ds = d \qquad .3.28$$

where d is given

The problem is then given as:

$$y' = \begin{cases} f(y,t), & t \in [a,b] \\ - & \\ f(y,t), & t \in [b,c] \\ + & \end{cases} \qquad 3.29$$

where b is defined by equations 3.27 and 3.28. This situation can be illustrated diagramatically as shown in Figure 3.1.

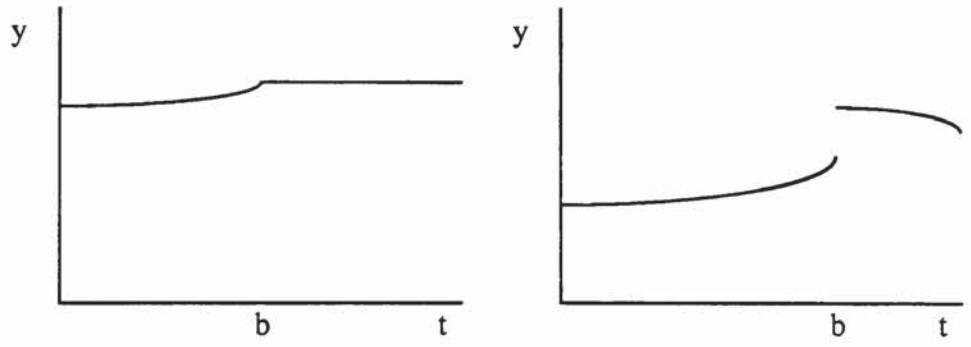


Figure 3.1. Discontinuity in the derivative of the state variable

In solving problems of this type, special measures must be taken to detect the discontinuity in advance, before it affects the smooth operation of the integration in several ways. For example, the algorithm will hunt around the discontinuity resulting in many unsuccessful steps being taken. The calculation of the actual switching time will also present a problem.

Various methods have been proposed in the literature for detecting the discontinuity and calculating the switching time (Carvar, 1978 ; Ellison, 1981). Carvar (1978) proposed the incorporation into the model of special discontinuity functions, which contain the

threshold information. As noted by Carvar himself, this procedure increases the size of the problem. Kuru (1981) proposed a method in which the point constraints are solved simultaneously with the state equations if the sign of the former have changed. This requires modifying a unit within the same simulation session in order to add the point constraints, which increases the model equations and variables.

Pantelides (1986) describes a "locking" mechanism used in SPEEDUP to determine state event time. The procedure is described as follows:

1 At the beginning of each step, the activation state of each discontinuity is determined. This is equivalent to checking if any of the threshold values have been crossed.

2 During the step, no changes in state are permitted even if one or more discontinuity conditions are activated. Thus the discontinuities are "locked" into their initial state, and consequently, the integrator is faced with a continuous system over the whole of the step.

3 At the end of the step, the discontinuity conditions are checked; if any of these are active, the exact earliest point of activation can be easily located.

This procedure may be seriously in error as it assumes that the function is continuous across the discontinuity. While this may be true for some state events, it will run into difficulty where the function is mathematically undefined at the point of the discontinuity, and this may cause some numerical problems.

Joglekar and Reklaitis (1984) proposed a strategy which is based on the use of the special facilities offered by the integration algorithm. The determination of discontinuity requires checking the values of the state variables (identified in an event list) to detect any threshold crossing and its direction. If the threshold values have been crossed for any

variables in the event list, then the switching time is determined by using the information stored in the integrator LSODAR (Petzold and Hindmarsh, 1982) to determine the switching time in a simple root finding procedure. This approach has the clear advantage that no extra equations are included in the model and the root finding procedure is not less accurate than the integration accuracy itself. Secondly it is more efficient in terms of computing time compared to the other methods mentioned above. However, it must be pointed out that this procedure is only applicable to methods which store the history of the integration in some form. DASP uses a similar method as described in Chapter 8, by modifying the integrator, DASSL (Petzold, 1983).

3.5 SOLUTION OF NONLINEAR ALGEBRAIC EQUATIONS

The nonlinear algebraic equations either generated by approximating the derivative terms in the differential-algebraic systems by a BDF method or by equating these derivatives to zero to obtain a steady state model can be represented as:

$$\mathbf{F}(\mathbf{X}) = \mathbf{0} \qquad . \ 3.30$$

where **X** is an N-vector of unknowns and **F** is an N-vector of functions defining the model equations. Many algorithms and software have been developed to solve these problems, as discussed in considerable detail by Sargent (1981). Those general purpose methods suitable for inclusion in an equation-oriented package can be classified as follows:

1 Locally convergent methods

These require good starting values to guarantee solution. Examples are the Newton-Raphson (NR) methods, and the quasi-Newton methods.

2 Methods with expanded region of convergence

These methods are usually hybrid, i.e. they combine two or more methods to expand their region of convergence. Examples include Powell's method (Powell, 1970) and modifications (Westerberg and Director, 1978; Chen and Stadtherr, 1981).

3 Globally convergent methods

In theory, these methods can converge practically from any starting values. Examples are the continuation methods (Sargent, 1981; Seader, 1985).

It is ironic that the most widely used methods by chemical engineers are the least reliable, but the most efficient when they are successful. These are the locally convergent methods like the Newton-type methods and the quasi-Newton methods.

The Newton-Raphson method first converts non-linear algebraic equations into their linearized equivalents. These are then solved iteratively using a linear equation solver, which can take advantage of the structure of the equations. The linearization results in a system of the form:

$$\mathbf{J}^m . \Delta\mathbf{X}^m = -\mathbf{F}^m \qquad ..3.31$$

where

$$\Delta\mathbf{X}^m = \mathbf{X}_{m+1} - \mathbf{X}_m$$

J is the Jacobian matrix, defined by:

$$\mathbf{J}_{ik} = \frac{\partial \mathbf{F}_i}{\partial \mathbf{X}_k} \qquad . 3.32$$

and m the iteration counter. The new values of the variables can be obtained from previous iterations as:

$$\mathbf{X}_{m+1} = \mathbf{X}_m + d\,\Delta\mathbf{X}^m \qquad . 3.33$$

where d = 1.

The Newton-Raphson method is said to have second-order convergence, i.e.

$$\lim_{m \to \infty} \frac{\| X_{m+1} - X^* \|}{\| X_m - X^* \|^2} = C \qquad . 3.34$$

where X^* is the solution, C is a constraint and $\|.\|$ is a suitable norm. This second order convergence means that close to the solution, the number of correct significant digits of the unknowns is multiplied by two in every iteration. The Newton-Raphson methods have the following drawbacks:

1. The matrix of partial derivatives must be evaluated in every iteration which is costly with regard to computer time.

2. A system of linear equations must be solved in every iteration.

3. Starting far from the solution, the method may diverge.

4. It cannot be used when the Jacobian matrix is singular.

The quasi-Newton methods do not require continual re-evaluation of the Jacobian matrix as the initial Jacobian is being updated at every iteration. The quasi-Newton methods have superlinear convergence properties, i.e.

$$\lim_{m \to \infty} \frac{\| X_{m+1} - X^* \|}{\| X_m - X^* \|} = 0 \qquad . 3.35$$

which means that the convergence is accelerated near the solution but slower than the Newton-Raphson method.

The methods with extended region of convergence were developed to overcome some of the problems mentioned with regard to the Newton-Raphson method (Shacham et al, 1982). They offer advantage over the NR method in the case of initial values far away from the solution but require more computer time. They have not been used in industrial simulators.

Methods with global convergence properties are available (Sargent, 1981). Although these algorithms are complicated, they usually guarantee a solution. For example, the continuation methods (Seader, 1985) transform the system of non-linear algebraic equations into a system of differential equations which can then be "integrated" using a suitable integrator. At each integration step, modified forms of the algebraic equations are solved using, for example, the Newton-Raphson method. Convergence will always be achieved by reducing the step size of integration, although this increases the computational effort required. It is envisaged that future development in the area of solving non-linear algebraic equations will move towards the use of global methods. However, the Newton-Raphson method is still the most popular, and all efforts are directed towards reducing its disadvantages as reported by Locke, (1981).

3.6 CONCLUSION

This chapter has briefly described the numerical methods and algorithms suitable for solving DAEs of the type generated in the dynamic simulation of chemical processes. DASSL (Petzold, 1983) is an integrator based on BDF which offers not only an efficient solution method for these problems, but can be adapted to handle discontinuities using the efficient method of Joglekar and Reklaitis (1984). It also accepts equations written in implicit form, making it the proper candidate for inclusion in an equation-oriented simulator using these strategies.

It has been shown that DAEs which describe dynamic simulation problems are generally

of the class of solvable types, or they can be transformed into such systems. This may require differentiation of the algebraic equations or reduction of the original system by eliminating some of the variables. Some of the circumstances in design which results in DAEs which are more difficult to solve can be avoided by detailed modelling and careful choice of design variables as is usually done in chemical engineering design.

Codes based on BDF which are used for solving DAEs (especially those formulated implicitly) require a consistent initial conditions for all the variables and derivatives to start the simulation. How this is done in DASP is described in Chapter 7.

CHAPTER 4

FEATURES OF DYNAMIC SIMULATORS

4.1 INTRODUCTION

This chapter discusses the main features of some representative dynamic simulators which use the equation-oriented approach or a variant. In Section 4.2, the basic features of an earlier attempt by Kocak (1980) are described with an assessment of the deficiencies. Section 4.3 discusses the structure of the CSSL-dynamic simulators. In Sections 4.4 and 4.5 the basic features of two of the available equation-oriented dynamic simulators, DPS (Wood et al, 1984) and SPEEDUP (Perkins and Sargent, 1982) are discussed. An analysis of the requirements for a new dynamic simulator is described in Section 4.6 and finally the computer environment is discussed in Section 4.7.

4.2 A DESCRIPTION OF ACES

4.2.1 MAIN FEATURES OF ACES

ACES, acronym for Aston Chemical Engineering Simulator, is described in the PhD thesis by Kocak (1980). It is a process dynamic simulator, designed for the simulation of the transients in start-up or after a throughput change in chemical plants. Its structure is based on a CSSL-type simulator called ICL SLAM (Anon, 1974), which is a modular package with the following main features.

1. It can simulate the dynamic behaviour of a continuous process using the modular approach.

2. There are no model equation routines included, rather the user must write a

program for his problem and use the simulator as an equation solver.

3. Other features include the use of argument list instead of Fortran common blocks to communicate state variables and their derivatives. The integrator is called as any other subroutine with no executive powers, and all the differential equations within a module are solved simultaneously.

All the models in ACES are derived from DYFLO (Franks, 1972). In order to avoid using outdated values of the state variables in the integration process and to ensure piecewise integration, ACES uses a feature known as "twice-round execution", where the model routine is called twice. In the first call, state variables are retrieved from the integrator to the model and local calculations are performed. The second call ensures that fresh derivatives are calculated and forwarded to the integrator for later simultaneous integration. The integrator used is DIFSUB (Gear, 1971b), which uses a variable-step variable-order BDF algorithm.

To evaluate ACES for this work, it was transferred from the ICL 1904S to HARRIS H500/H800 mainframe and ICL PERQ computers. After some minor modifications, it was possible to benchmark some of the examples from the ICL 1904S given by Kocak (1980) on the versions of the package on the other two computers. The package showed a high degree of portability and robustness, giving the same results on the three computers. An attempt was made to interface the GKS graphics package on the ICL PERQ to ACES. This was discontinued partly because the GKS implementation was not reliable, and partly because ACES was deficient in several respects, as described below.

4.2.2 DEFICIENCIES OF ACES

ACES was an attempt to overcome the problems of sequential modular simulation in DYFLO, but it did not go far enough. In the author's opinion, ACES was out-of-date by the time it was completed. Some of the main deficiencies are:

1. The user has to be highly involved in any simulation session, since a main program must be written, which must call the modules from ACES library routines, knowing the types and functions of all the common blocks in the package. This type of "main program" approach inconveniences the user, who must know a great deal to order the calls correctly.

2. The documentation is inadequate. The thesis is full of unnecessary jargons, the procedure for writing a new module is not properly defined and most variables in common blocks and their functions are not well documented. A manual could have solved these problems.

3. The integrator, DIFSUB (Gear, 1971b), requires the equations to be written in standard form. All algebraic equations must be solved locally within the module where they occur. This makes it impossible to use the same model for both steady state and dynamic simulation. As described in Chapter 3, DIFSUB cannot handle some stiff problems, which other integrators like EPISODE can solve easily.

4. ACES cannot handle design and optimization efficiently. The feature RERUN, provided in the thesis is the same as the "controlled simulation" of the sequential modular approach, as discussed in Chapter 2.

5. The variables in common blocks are liable to corruption, since they must be included in the user-written subroutine, USMLND, and the user could therefore access them directly.

6. Although the trapping of error is good, the messages reported are incomprehensible without access to the program listings.

7. ACES lacked most of the features of structured programming, making it difficult to debug and extend.

Therefore, although Kocak (1980) strongly criticized DYFLO's sequential modular approach, ACES cannot claim to be truly an equation-solving package, which it was intended to be, as it cannot claim the advantages associated with this approach, although it did solve the simultaneity problem in DYFLO (Franks 1972).

4.3 THE CSSLs

A CSSL (acronym for Continuous System Simulation Language) is an application-oriented system for investigating the dynamic behaviour of physical systems described by sets of differential equations. The design of a CSSL follows the guidelines laid down by the CSSL Committee (Anon, 1967), although since then a lot of changes have been introduced to enhance present day CSSLs. The basic principles can be summarised as follows:

1 Definitive Structure

The program has clear cut divisions with specific functions. These divisions are the Initial, Dynamic and Terminal Regions, as shown in fig 4.1. The Initial region contains those statements which are executed once at the beginning, for example parameter declarations and initialization of variables. The Dynamic region is subdivided into two sections, the derivative and parallel sections. The derivative section contains statements for calculating the derivatives of the state variables for the integration routine while the parallel section provides for statements which need to be executed prior to output of variable values. The Terminal region contains statements which determine the termination of the simulation run.

2 Automatic Sorting

The model definition statements may be written in any order and will be automatically sorted into a correct execution sequence, a feature which assists the modularization of the

code.

3 Provision of Different Integration Routines

In all CSSLs, a variety of integration schemes are provided. Although earlier versions had only integration routines using explicit methods, the current versions provide for integration routines for solving stiff systems.

4 Multi-derivative Capability

This is a feature which makes it possible to use different step lengths of integration at different intervals.

```
Program
    Initial
        Code executed at beginning
    End of Initial
    Dynamic
        Derivative
            Code describes state variable derivatives
        End of Derivative
        Code executed at communication time
    End of Dynamic
    Terminal
        Code executed at end of run
    End of Terminal
End of Program
```

Figure 4.1 - The structure of a typical CSSL

5 A Library of Functions

Most CSSLs provide Fortran-like functions or subroutines which can be used to do specific functions such as summation. These functions are usually compatible with Fortran library routines and the user can provide his own function library.

Other features are usually present such as interaction and graphics, error diagnostics capability, macro processor and macro library, which provide extensions to the language. An example of a CSSL is ACSL (Mitchell, 1978). The user usually describes his problem in this language which is preprocessed to produce a Fortran code. This is linked to a library of Fortran subroutines which are then linked and executed. However, high level languages other than Fortan are also used.

6 Novelties and Deficiencies

The features of a CSSL provide the capabilities most users need, who are not interested in the numerical aspect of dynamic simulation, and who want programming tools that speed up program development. The structuring of the language and the use of macros make it easier to describe a problem for solution. However, as pointed out in Chapter 1, CSSLs do not cater for the needs of chemical plant simulation. New attempts to include chemical plant models into the library of functions may look attractive, but they fall short of the basic requirements of a process engineer. CSSLs do not handle implicitly defined differential-algebraic equations. In fact, most of them do not handle algebraic loops at all (Wood et al, 1984). Others don't have chemical plant models. Thus CSSLs can best be regarded as a means of solving the users problem who must describe his problem in explicitly defined differential equations. However, the structure of the CSSL can provide an alternative structure for solving the complexity problem of an equation-oriented dynamic simulator.

4.4 DPS

(a) General Design

DPS (Thambynayagom et al. 1981 ; Wood et al, 1984), derived from the steady state counterpart, JUSE-L-GIFS (Anon, 1970), is a modularly organised CSSL-type simulator written in Fortran. Both material and information flow steams are allowed. It

can handle both steady state and dynamic simulation. DPS can solve a variety of discrete events such as switching of flows, starting and stopping pumps and opening and closing of valves, all of which can be state or time events. All these features can be combined to study start-up and shut-down procedures and determine the effect of process upsets on throughput, recovery, conversion and energy efficiency.

(b) Data Input

Data input is interactive, in which a user responds to prompts on the terminal. This could be time consuming in the case of a realistic chemical plant.

(c) Model Equations

DPS is a CSSL-type dynamic simulator with the added feature of a library of model routines of chemical plants. A user can use modules from the DPS library, which are made up of DPS standard functions (eg. summation, step, pulse and some models of chemical plants). These can be combined to transform a flowsheet into an equivalent block diagram. It is also possible for a user to add his own routines. A distinction is made between the algebraic and differential equations and they are handled by different methods.

(d) Method of Computation

The differential equations are converted to difference equations by a selected explicit integration method, forming a system of algebraic equations which has to be solved at every step. These equations are represented by a non-linear signal flow graph, which shows the functional dependence among the variable types in the system (eg. component flows, temperature, pressure). This signal flow or directed graph is then automatically analysed to produce an efficient, ordered computational sequence for the algebraic system representing the simulation. The integrators are fixed step length codes with no error control.

(e) Novelties and Deficiencies

DPS is an attempt to combine the best features of a CSSL type simulator, discrete system simulator and a process simulator in one. However, with explicit integration methods and no error control, it is not possible to monitor integration accuracy. The structure of the algebraic system generated depends on the chosen integration algorithm and the digraph representation has to be changed when a new integrator is chosen. Moreover, it has a complex definition of variable types. DPS cannot handle implicitly defined differential-algebraic system. In fact, the differential and algebraic equations are handled differently.

4.5 SPEEDUP

(a) General Design

SPEEDUP (Perkins and Sargent 1982) is also derived from its steady state counterpart SPEEDUP (Sargent and Westerberg, 1964). Unlike DPS it is written in PASCAL for ease of production, maintenance and portability. However, the system produces executable Fortran code from the input data which is then linked with the library of Fortran numerical routines prior to execution. SPEEDUP permits steady state and dynamic simulation and optimization of a chemical plant. It can solve differential-algebraic equations, in which the differential equations are written in standard form.

(b) Data Input

SPEEDUP is a data-based system, in which a special engineering-oriented language is used to input information and commands. The system interfaces with a databank containing plant unit models, physical properties, cost data and Fortran subroutines of procedures for calculating vapour-liquid equilibrium and enthalpy coefficients. It can be run either batchwise or interactively.

(c) Model Equations

SPEEDUP contains a number of unit operation models which are programmed as differential-algebraic systems. A user can combine them to simulate his problem. New model routines can be added if they are not available in the SPEEDUP library of model routines. These must be written in a special language. The system also allows the use of procedures so that subroutines of the types written for the sequential modular simulators can be used.

(d) Method of Computation

The model equations are a system of differential-algebraic equations which are solved directly using a suitable implicit DAE solver. It is possible to change the solution method during a simulation session depending on the characteristics of the equations. However, the DAE solver can only handle systems of equations of the diagonally implicit type as described in Chapter 2.

(e) Novelties and Deficiencies

SPEEDUP is a state-of-the-art equation-oriented process dynamic simulator. The steady state option is more developed than the dynamic counterpart, and the latter has benefitted greatly from the features of the former. Because it is databased, the ability to manipulate data and text is greatly enhanced. However, the method used in SPEEDUP to handle state events may result in numerical problems if the model is mathematically undefined after the point of discontinuity as discussed in Section 3.4 of Chapter 3. SPEEDUP is written for a mainframe and runs on it. It cannot make use of the increasingly very powerful and cheap microcomputers, which are readily available.

4.6 REQUIREMENTS FOR DASP

The requirements for a general purpose dynamic simulator are numerous and can be

outlined by describing the basic design elements (Heyt and Fairchild, 1982). Equation-oriented dynamic simulators can be thought of as comprising the following elements:

1. The Executive routines.
2. The Input/Output routines.
3. The Equation Generation package.
4. The Equation Solving package.
5. The Model Library.
6. The Event Processing package.
7. The Thermodynamic Interface routines.

1. The Executive Routine

The Executive should supervise and co-ordinate all the functions of the other routines that make up the package. To do this effectively, a suitable structure of the package must be designed, which should be as simple and flexible as possible in order to be easy to debug and extend. The CSSL-type structure is very attractive this can be adapted and extended to cater for the complexity required of a modularly-organised equation oriented dynamic simulator.

2. The Input and Output Routines

The Input and Output features should be tied to a database. This will enhance input of data and information via a command language as well as output of data, writing of reports for the user, general communication between the user and the system and as a store of data necessary for modifying the flowsheet of the plant interactively. Some of the data items needed to start the system are the process topology, components data and the initial values of variables and parameters. Interactive input and output are basic features to be included. For example, with graphical input of the above data, a review of calculation as it proceeds, the making of decisions based on preliminary results and quick correction of mistakes and faulty assumptions, are easily achieved in a completely

interactive simulator.

3. Equation Generation

The Equation Generation package assembles all the equations to be solved from the model library. This should be done without requiring too much overhead and in a consistent manner to form a well determined system. Therefore, a mechanism needs to be developed to carry this out and this could be very difficult to achieve. Also, an option should be provided for the user to use his own model equations via an interface routine.

4. Solution of Model Equations

Dynamic simulation of chemical plant is about solving the dynamic models of the problem, so the equation-solving routines are at the core of the package. As shown in Chapters 2 and 3, the equations to be solved form a differential-algebraic system. The numerical methods used should be able to handle this. To provide initial values for dynamic simulation, the steady state values can be perturbed. This requires the solution of the system of non-linear algebraic equations of the steady state model. Simultaneous solution of the model generates the need to solve large numbers of equation which are very sparse in the variables. A suitable sparse matrix package must be used to reduce the required computer memory.

5. The Model Library

The model library contains equations for the different unit operations. This relieves the user of considerable programming effort. A module should be easy to write, with a general format. It is possible to use the nonlinear equations or a linearized form of them. However, the strategy used in formulating the model equations should be flexible enough so that the same equations could serve several purposes such as steady state and dynamic simulations, as shown in Chapter 2.

6. Event Processing

State and time events frequently occur in dynamic simulation and the ability to process

them is expected of any dynamic simulation package. The main problems occur with regard to the processing of state events, which is analogous to integration over a discontinuity as described in Chapter 3. The choice and implementation of a suitable algorithm to achieve this must be made.

7. The Thermodynamic Interface Routines

It has been estimated (Cameron, 1983) that almost 90% of the execution time is spent in processing component data, computing and correlating thermophysical constants and calculating the vapour-liquid equilibrium coefficients and enthalpies of the components in the system. This has to be born in mind when designing the thermodynamic interface routines. There are a number of possibilities to reduce the computation time, for example, the use of polynomial approximations instead of the rigorous thermodynamic equations. However, for flexibility in interfacing with most already existing packages, it is better to see it as independent of the model equations. Also a databank or a number of alternative banks of physical properties will be needed. This should be as comprehensive as possible for general work.

4.7 THE COMPUTER ENVIRONMENT

Recent developments in computer technology have made it possible to develop serious chemical engineering programs on microcomputers. The low cost of these systems, the large memory which is now available and the ability to provide an inexpensive conversational environment for the engineer are some of the factors that have made this possible. More and more software previously run on mainframe computers is now being converted for use on microcomputers. The friendly environment which microcomputers provide, includes input and output which can be linked to word processors and spreadsheets for both tabular and graphical input/output. In one case an interactive design program, when moved to a microcomputer, actually used less total

computing resources, and only 25% of the computing time for engineering calculation of its batch counterpart (Cameron 1983).

For an interactive program to be most useful, careful consideration must be given to the way it communicates with the user. A specific command language or a menu-based system can be used. A variation of the menu-based system is the mouse-driven system, in which a pointing device, the mouse, is used to simply point at and select the menu item. Almost all chemical engineering programs are written in Fortran language. Although it lacks most of the advantages of structured programming, engineers are used to this language and Fortran 77 offers better features than the previous versions. Also Fortran 77 compilers are available for almost all business micros. The capabilities of the computers to be used are described in the next section.

4.7.1 ICL PERQ

The ICL PERQ, located at the Chemical Engineering Department, is a single-user computer with a large internal storage capacity (24 MByte Winchester hard disc, 512 KB of main memory and 1 MByte floppy disc drive). It has a 16 bit micro-programmed processor, a high resolution graphics using a 1024 x 768 bit mapped raster display, a standard keyboard with special function keys. Two RS232 C interfaces for communication with serial devices such as some printers or communication on wide area networks to other devices and computers are also provided. A GPIB interface (IEEE-48standard) for communicating with parallel devices such as laboratory instruments and some printers and a high resolution graphics tablet equipped with a four button pointing device which can be used to input or construct diagrams or point to parts of the display area are provided.

PERQ uses a version of the UNIX operating system, called PNX. UNIX was originally developed by Bell Laboratories, for program development in a research environment. Most of UNIX and its software are written in the C language. PNX on the PERQ has all the well-known features of UNIX, plus the advantage of running on a powerful graphics

oriented computer. One of the main features of UNIX and PNX is the multitasking capability. This has been enhanced by PNX with the addition of the following features.

1 Window Management System

This exploits the high resolution graphics display and enables one to take advantage of the multiprocessing ability of UNIX. Thus many windows can be created, either directly or via programs, with each window acting as another terminal as in multiuser systems. The Window Management System also includes a pop-up menu package so that pop-up menus can be easily incorporated into programs.

2 Mixed Language Programming

PNX provides Fortran 77, C and Pascal as well as Assembler languages. Mixed language programming is acceptable to the available compilers, whereby programs written in different languages can be linked together and run.

These features could have provided an suitable environment for development of a dynamic simulation package, but for the following drawbacks:

1. The graphics package, GKS, provided was not reliable. Also only the primitive routines are available in the package, thus requiring enormous work for the user to develop graphical applications.

2. No printer or plotter was available for the PERQ.

3. System dependent software written for the PERQ will not run on any other non-compatible computer. This will limit its usefulness.

4. The system was very slow to compile, a small Fortran program takes several minutes. Also, the available compilers impose several restrictions as regards the length of Fortran expressions.

For these reasons, the program was developed using only standard Fortran 77 on the PERQ, which was then transfered to the IBM PC AT.

4.8 CONCLUSION

The original objective of this work was to enhance the capabilities of ACES, especially in the areas of interaction and graphics input of data and output of results, extend the module library and include a VLE and enthalpy calculation package. However, after a very careful survey of the literature on the state-of-the-art, methodology and future trends in dynamic simulation and a critical evaluation of ACES and the design features of other simulators, it was decided to start on a clean sheet instead of doing patch-work on ACES with loss of efficiency. However, there are some design features of ACES which can be tailored towards the design of DASP. For example, the CSSL structure can be adapted for use in a truely equation-oriented environment. The design features of DASP are described in the next chapter.

CHAPTER 5

THE DESIGN OF DASP

5.1 THE DESIGN PHILOSOPHY

DASP is acronym for Dynamic Analysis and Simulation Package and a product of the work described in this thesis. The design philosophy used in DASP arises from the work already described. The literature survey presented in Chapters 2, 3 and 4 described the state of the art and a critical analysis of the issues involved in dynamic simulation of chemical processes and also raised some of the problems which need to be tackled in order to design a new dynamic simulator. In Chapter 2, it was concluded that a modularly-organized equation-oriented approach is best suited to solve most of the problems of the sequential modular approach. Also the simultaneous solution of the differential-algebraic equations of the model has several advantages. These include the use of the same equations for both steady state and dynamic simulations and making of all the variables in the model "available" for global manipulation. Also design and optimisation calculations can be carried out much more easily. However the implementation of a general purpose equation-oriented approach is not easy, compared to the design of a sequential modular approach (Hlavacek, 1977 ; Perkins, 1984). This is mainly due to the complexity needed in the executive routine. In an equation-oriented simulator, the user's problem is very easily described by systems of equations, but this description does not give any details about the procedure for obtaining a solution, a job left to the executive program to determine. This job becomes much more difficult if the equations to be solved are to be generated from in-built model routines. The design of the executive program can also determine the structure and working of other subroutines in the package. Therefore, an integrated approach was used to design DASP.

The special purpose subroutines which implement the features of the package are

designed in such a way that given certain inputs these routines calculate the required outputs. Moreover, since certain simulation options require the services of a number of these features, a co-ordinating routine is needed for each simulation option, which will organize the calls to the special-purpose routines so that the job required of the option is fulfilled. In DASP these co-ordinating routines are the interface routines for dynamic and steady state simulations, event processing, and the modify routine. This design reduces the workload of the executive routine, which will now call the interface routines as required. The executive routine is also organized into regions, each region representing the simulation option as given before. It becomes easy to add new options and to debug the program as everything about an option is concentrated in one region only.

The next problem addressed is how to design the model routines so that it is easy to assemble the equations to be solved from these model routines, and transfer variable values between modules. It is desirable for all model routines to have a common design structure. This has the clear advantage that it is easy to add new model routines. Also if a method is developed for assembling the equations from one module, this same method can be used for any other module. This has led to the development of the unit module, which represents a whole or part of a unit operation, in which only one occurrence of a variable is permitted. Thus if temperature is regarded as a variable, in a unit module only one temperature variable is allowed to occur. If there is more than one temperature variable in a unit operation, then that unit operation will be made up of more than one unit module. Also, all variable types are given unique code numbers, which can be used to identify them in a module. A module is functionally divided into initial, function evaluation, Jacobian evaluation, output, event code and terminal sections and each section is identified by the value of a control variable. This, with the module structure is all that is required to assemble the equations to be solved from the modules. Details of the structure are discussed in Chapter 6.

In order to use the same equations for both steady state and dynamic simulations and possibly optimization, it is necessary to use a standard format for formulation of these equations in the modules. Some simulators introduce new variables and new equations for every differential equation in the model as discussed in Chapters 2 and 3. DASP uses an implicit formulation strategy which does not require introduction of new variables or equations and reorganisation of these equations for any type of simulation as discussed in Chapter 2. This is described in Chapters 6 and 7.

The software development philosophy was aimed at portability, robustness, ease of debugging and ease of future development. Firstly, different channel numbers are used for the different input and output operations which is useful in redirecting outputs to files or to different windows in an environment with window management facilities. Secondly, all the variables and parameters and their code numbers are stored in a single array by allocating spaces or beads to them according to their length. The array for integer values is ICORE (*) while that for floating point values is CORE (*). Also workspaces are allocated to other arrays used in the numerical calculations from these CORE (*) and ICORE (*) vectors. This reduces the size of the program to the number of variables in the system to be solved. Thirdly, only standard features of Fortran 77 were used throughout, which will enhance portability.

The event processing philosophy is based on the assumption that the user knows the sequence of occurrence of these events, both state and time events. Also only single events are allowed. Although this may not be satisfactory for all purposes, it reduces the time necessary for searching the event list at every step. One other feature of the event processing routines is the ability to modify a flowsheet during the simulation. This is a useful feature in start-up and shut-down of chemical processes and in batch operations.

The numerical solution algorithms used include the use of the state of the art software, DASSL (Petzold, 1983), which was modified and enhanced, and also the development

of a method for the initialization of differential-algebraic equations.

Thus although DASP has a simple design which was the clear objective right from the onset, the implementation of this design ensures the effective performance of the necessary work. In the remainder of this chapter, we describe the main features and the structure of DASP and the working of its various regions. The other parts of the package are grouped under the headings of equation generation package (Chapter 6), equation solving package (Chapter 7) and the event processing package (Chapter 8). In these chapters, only the overall strategy is discussed. Detailed implementation can be seen in the appendices and the listing of the programs.

5.2 THE MAIN FEATURES OF DASP

DASP is written in standard Fortran 77 and was originally implemented on an ICL PERQ computer. It has later been transferred to an IBM PC AT. The program has been successfully run on the ICL PERQ, IBM PC AT, HARRIS H500/H800 and on VAX CLUSTER with very little modification, giving the same results for the same examples. The only system dependent features are the default unit numbers for the standard input and output, and a subroutine, CLEAR, which clears the screen after every input. Although this subroutine is written in Fortran, it requires the ASCII codes that clear the screen for the particular computer system. The two routines, CLEAR and a BLOCK DATA routine which defaults the values of all variables and parameters and provides the standard input and output unit numbers have been separated from the rest so that they can easily be modified and reloaded during installation. DASP is therefore highly portable. In implementing DASP, consideration was given to the discussion presented in Chapter One to Chapter Four and the design philosophy of section 5.1. The main objective was to use recent design methodologies and numerical algorithms and combine them with the capabilities of Fortran 77 and computer power into a system that works, is portable and robust. An earlier implementation of this package has been described in

Fletcher and Ogbonda (1985). The present implementation and the main features of DASP have been briefly described in Fletcher and Ogbonda (1987). They include the following:

1. It is an equation-oriented process dynamic simulator in which the model equations describing a chemical plant can be assembled from in-built library routines of whole or part of a unit operation model. This is the modularly organized equation-oriented approach as described in Section 2.3.4.

2. It is semi-interactive, i.e. the bulk of the input data is read from pre-prepared data files, while some control parameters and all communication with the user are via menu-driven prompts on the terminal.

3. All the dynamic model equations, both differential and algebraic are solved simultaneously using an advanced mixed equation solver, DASSL (Petzold,1983), which does not impose any restriction on the formulation of the model equations. DASSL is described in Chapter 7.

4. The system to be solved can have two types of Jacobian matrix structure, namely dense and sparse matrix. To handle the sparse option, a sparse matrix package, MA28 (Duff, 1977) was interfaced with it.

5. The model routines have a common structure, making it easy to write new modules. The order of writing the model equations is at the discretion of the developer. The user can introduce his own model routines via an interface routine, USRSUB as explained in Appendix B4.

6. The system can handle both time and state events in an efficient way, a feature which is useful in simulating the start-up and shut-down of chemical plants, and in

batch operations. Event processing is discussed in Chapter 8.

7. The user has access to all the variables, parameters, control information and problem descriptions during the simulation. The data can be viewed and modified.

8. The steady state solution of the problem can be obtained if required, by the use of either the Newton-Raphson method or Broyden's method. This does not require any reorganization of the model equations or the introduction of new variables and equations.

9. Vapour-liquid equilibrium and enthalpy calculations are carried out using the UNIQUAC (Prausnitz et al, 1980) or UNIFAC (Fredenslund et al, 1977) programs although user-supplied thermodynamic models can also be used.

10. DASP runs on a micro computer with 640 KByte of memory. It is a portable and robust program.

11. DASP can be restarted or rerun from any point within the interval of integration using either the initial values or any of the intermediate values. It is also possible to run several cases, one after the other, with data from the same data files.

5.3 THE STRUCTURE OF DASP

The Executive program, DASPM, is the main driver routine of the package which supervises, directs and controls the functions of the components described starting from Chapter 5. The structure of DASPM is a modified CSSL structure, which was discussed in Chapter 4. However, the actual implementation and the functions of the

regions are totally different. Also the regions are further subdivided into sections. These regions, identified by the value of a control variable, JSECTN, are as follows:

1. INITIAL REGION, typified by JSECTN = 0
2. DYNAMIC RECION, which is further sub-divided into sections as follows:
   (a) steady state section (JSECTN = 1)
   (b) dynamic simulation section (JSECTN = 2)
   (c) perturbation section (JSECTN = 3)
   (d) event section (JSECTN = 4)
3. TERMINAL REGION, which has section as:
   (a) exit/restart/rerun section (JSECTN = 5)
   (b) error analysis section (JSECTN = 6)

DASPM controls the working of these regions. It is called by the user written main program as:

CALL DASPM (USRSUB, CORE, LCMAX, ICORE, LICMAX, IFLAG)

where

USRSUB is the dummy name within DASPM for the user-written subroutine, which contains equations to be solved as explained in Appendix B4.

CORE (*) is a real workspace

LCMAX is the length of CORE (*)

ICORE (*) is an integer workspace

LICMAX is the length of ICORE (*)

IFLAG is an integer flag, set to a negative value if any error condition occurred during the simulation.

A typical main program written by the user is as follows:

```
PROGRAM PROBLEM
EXTERNAL USRSUB
PARAMETER ( LCMAX=2000 , LICMAX=1000 )
DOUBLE PRECISION CORE (LCMAX)
INTEGER ICORE (LICMAX) , LCMAX , LICMAX , IFLAG
IFLAG = 0
CALL DASPM (USRSUB , CORE , LCMAX , ICORE , LICMAX , IFLAG)
STOP
END
```

A schematic representation of the overall structure is shown in Fig. 5.1. In the sections that follow, a description of the logic of regions is given. It should be noted that this type of structuring helps not only to make the program easier to understand and debug, but also makes it possible to add new regions with new JSECTN values, for example, an optimization region.

5.4 THE INITIAL REGION

When DASPM is called, all the variables, parameters and control information are automatically defaulted via a BLOCK DATA sub-program. The arrays that store the locations of variables and parameters are initialized by calling the INIT subroutine. A file, MESFLE, which contains messages describing error types, units and names of variables and parameters, is opened for direct access input transfer. The subroutine, INPUT1 is called to read all input data from files and terminals.

Input data to DASP is via data files prepared by the user, details of which are described in Appendix B2. However, the general principles are given below. In general, the following files may be needed:

1. CINDAT, general input data file
2. CUNIT, initial values and error tolerances data file
3. CTOPOL, process topology data file (optional)

4. COMDT, components data file (optional)

The names of the data files given above are their dummy names. The user is required to supply their real names.

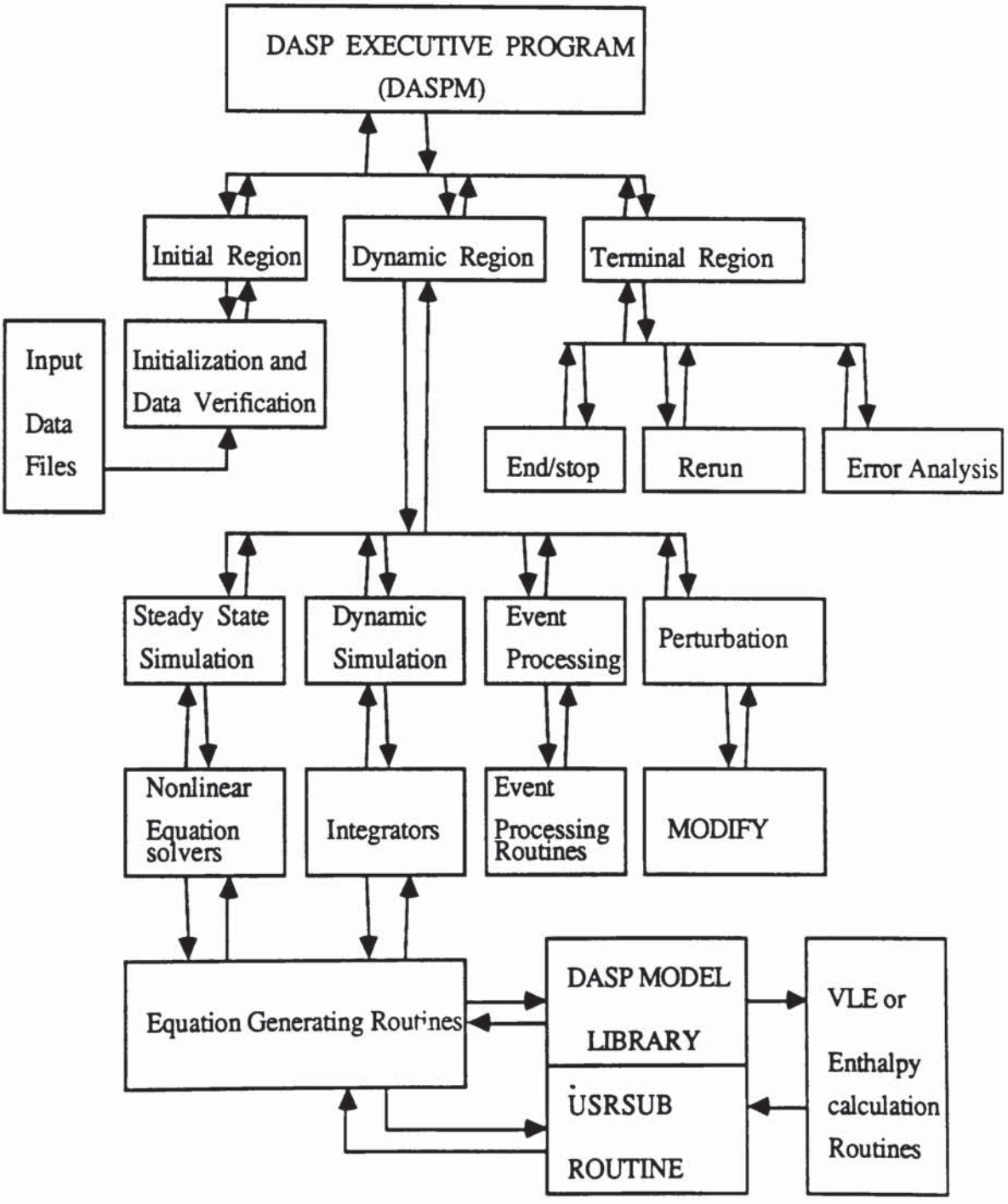


Fig 5.1 Overall Structure of DASP

In CINDAT, the user supplies in a specified order the values of all control parameters, integration parameters, output and plotting data and information about the Jacobian matrix of the system.

CUNIT gives all the information about the module parameters, both integer and real types, and the initial values of variables and derivatives in a module. Also the values of the error tolerances of the equations, relative and absolute, are given if the vector error tolerance option is chosen. In general, the following data are supplied.

1. Integer parameters, which give information about the model and parameter types selected etc.
2. Real parameters, which give data for all parameters in the module.
3. Initial values of the variables.
4. Initial derivatives of the variables.

In CTOPOL, information about the connections of the units to one another is given. These include:

1. Unit data, for example the number of units, unit identification numbers and unit codes.
2. Stream information, for example the number of streams, stream labels, units of origin and units of termination.
3. Connectivity matrix, for example, the number of streams connected to a unit and their units of connection.

COMDT is the component data file, which gives the information about the number of unique components in the system, their identification numbers (starting from 1) and method flag for VLE and enthalpy calculations and the relevant data.

Data files CINDAT and CUNIT are always required, while CTOPOL is only needed if KMODEL, the source of model flag, equals 0 or 1. COMDT will be needed if there are chemical species in the system.

As input is semi-interactive, the user responds to prompts on the terminal to read these data files, and to direct or interrupt the simulation. All prompts are menu-driven. All data input are controlled by the subroutine, INPUT1, which calls the relevant routines to read the initial values, process topology etc. This makes it easy to replace the input routines with new ones when necessary. Also, multiple simulation runs can be carried out by providing the relevant data in the input data files in order. If any errors occur during this input stage an appropriate message is displayed and execution is stopped. Otherwise, the work file WKFILE is opened for formatted sequential access. All the variables in the system to be solved are first written to this file after each iteration or step. In order for a selected number of variables to be read form the WKFILE file and written to the results file, WKFILE is closed and then opened for direct access. This approach is also used to select the results of a particular step to use in the restart option. Before leaving this section, the MODIFY routine is called, so that the user has access to all the variables and parameters, as described in Chapter 8. Then all the initial values are written to the work file, and control is passed to the dynamic region.

5.5 THE DYNAMIC REGION

This section controls all aspects of the numerical calculations, including dynamic simulation, steady state simulation, perturbation and event processing. The value of the control variable, MODE, determines whether dynamic or steady state simulation or both are to be carried out while JSECTN determines the logic of execution, as described in Section 5.3. If breakpoints have been set, at the appropriate time, control is passed to the user via the MODIFY routine. Unless the program is interrupted, the program jumps to the appropriate section according to the value of JSECTN.

5.5.1 STEADY STATE SIMULATION SECTION

This section handles the interface to the non-linear equation solvers by calling the subroutine ASSUB as:

CALL ASSUB (USRSUB, CORE, ICORE, IFLAG)

The ASSUB routine initializes all control information, allocates storage areas and checks all input data necessary for solving the non-linear algebraic equations at the first call (JSTART = 0). Then either NRSUB, BROYDN or ALGSUB is called depending on the value of MFALG, to do one or more iterations on the variables depending on the output or break flag options set, before control is returned to DASPM. The performance flag, IFLAG is then tested for any errors in which case control is passed to the error analysis section of the Terminal Region. This procedure is repeated until the solution is found or the maximum number of iterations exceeded without a solution. The variable values are then written to the work file and control is passed to the terminal region.

5.5.2 DYNAMIC SIMULATION SECTION

The sub-routines that solve the differential-algebraic equations of a chemical plant are controlled by this section via the DERIV interface routine. At the first call, DERIV initializes all control information, allocates storage space, and verifies input data relevant to dynamic simulation for the specific DAE solver selected. Then DASSL or DYNSUB is called to integrate from the current time, TIME to the next output time, TOUT, as specified by the user. On return, either successful or unsuccessful control is passed to DASPM. On each call to SDASSL or DYNSUB, the value of TOUT is calculated and this value is tested against:

1. TFIN, so that TOUT ≤ TFIN
2. TEV, so that TOUT ≤ TEV

3. TDISCN, so that TOUT ≤ TDISCN

where

TFIN is the final time of integration

TEV is the next time when time event will occur (LEVENT > 0)

TDISCN is the time when there is a discontinuity in the model (IDISCN = 1)

In DASPM, if any errors occur, control is passed to the error analysis section. Otherwise the OUTPUT subroutine is called to write the current values of the variables into the work file. If plotting of variables is requested after every step, LPLOT = 1, then PLTSUB is called to supervise the plotting. If an event has occurred, LEVEND = TRUE, then control is passed to the event processing section. If IDISCN = 1 and TOUT = TDISCN, then a message indicating that there is a discontinuity in the model at the current time is displayed and the program is terminated. If the upper limit of integration has been reached, TOUT = TFIN, then control is passed to the terminal region. Otherwise DERIV is called again to continue the integration, via the statement which directs JSECTN to jump to the relevant section.

5.5.3 PERTURBATION SECTION

In this section, the MODIFY routine is called to perturb any variables or parameters in the model. This can be done via the variable or upset option as described in Chapter 8.

5.5.4 EVENT PROCESSING SECTION

This section handles the calls to STESUB, which is the interface to the event processing routines. If any errors occur during this process, the error analysis section is called, otherwise, LEVEND is set to FALSE and control passed to statement number 800. Details of event processing are described in Chapter 8.

5.6 TERMINAL REGION

5.6.1 END SIMULATION/RERUN SECTION

The terminal routine, TMNSUB, handles all aspects of normal termination of the program as well as rerun/restart of simulation. It is called as:

CALL TMNSUB (USRSUB, JS, IFLAG, NRUN, CORE, ICORE)

where

JS is an integer variable, which is set as follows:

0 = terminate simulation

1 = reinitialize simulation

2 = continue simulation from where it was stopped

3 = run a new system with data in the same files.

NRUN is the number of reruns or restarts that has taken place so far.

On entering this section, the following message is displayed:

** TERMINAL REGION **

** TERMINAL MENU **

0 = Exit/End simulation

1 = Rerun present system after perturbation

2 = Run a new system with data in same files

3 = Run a new system using new data files

4 = call the MODIFY routine

** Select an Option **

These options are explained below.

1 Exit/End Simulation Option

This option terminates the simulation. Normally, the input data, and the specified output variables are read from the work file, and written to the user result file. Then all open

files are closed before exit from this routine.

2. Rerun Present System after Pertubation

In this option, the following message is displayed:

** Select an Option as Follows: **

0 = Rerun using the initial values

1 = Rerun using latest variable values

2 = Rerun using the i^{th} variable value,

Whichever option is selected, the relevant values are read from the work file into the variables and derivatives storage space, XVAR(*) and DXVAR(*). The MODIFY routine is then called after which control is passed to DASPM for a rerun.

3 Run a New System with Data in Same Files

Here the results of the previous run are written to the result file. All open files are not closed and control is passed to DASPM for initialization.

4 Run a New System using New Data Files

The procedure is the same as in item 3 above except that all input files are closed before exit from this routine for reinitialization.

5.6.2 ERROR ANALYSIS SECTION

Wherever an error occurs the routine, REPORT is called to write messages which describe the cause of the error. This routine uses the error code returned from the subroutine where the error occurred and writes further messages to an output file designated for errors. Although certain error types can be corrected so that simulation can continue, the present version terminates the execution of the program after writing the messages. The error messages are of a standard length, stored in a text file. This

can be read by random access when necessary. This avoids including the messages in the program text, and makes modification easy.

5.7 CONCLUSION

This chapter presented the design philosophy, the main features and the structure of the DASP program. It has shown the functional divisions of the executive program, DASPM, which controls the working of the other parts of the package, via the interface routines called at these regions. This type of structure was found to be very effective in reducing the complexity required of an equation-oriented executive to be able to carry out its function properly. As a result, this type of approach was used in the design of the structure of the model routines descussed in the next chapter.

CHAPTER 6

EQUATION GENERATION

6.1 INTRODUCTION

The equation generating routines assemble all the model equations for a user's problem from those modules in the DASP library which correspond to the unit operations in the users flowsheet or from the user-written model routine, USRSUB. All the modules are designed to have a common structure. All the equations are written in the form:

$$F(X) = 0 \qquad \text{..... 6.1}$$

Thus every function evaluation requires the calculation of the residuals of the equations irrespective of the type of equations. A differential-algebraic system of the form:

$$y' = z^2 + 2y$$
$$0 = 2z + y^2 \qquad \text{..... 6.2}$$

will be written in the form

$$f_1 = z^2 + 2y - y'$$
$$f_2 = 2z + y^2 \qquad \text{..... 6.3}$$

The structuring of the equations in this form is made possible by the equation solving routines, which require only the residuals of the equations during function evaluations. In this chapter, DASP's terminology and the structure of a model routine are described. It is shown how DASP assembles the equations from these modules or from the

USRSUB routine and how variables are passed between modules.

6.2 DASP's BASIC TERMINOLOGY

In order to describe this work clearly, a brief definition of the terminology used to describe DASP is given. Only some of these terms are defined as others will be described in the context where they will be most understood.

6.2.1 VARIABLES AND PARAMETERS

The distinction between variables and parameters is flexible, depending on the context and modelling assumptions made. A variable is any intensive or extensive characteristic of a system, whose values are to be determined. In the case of dynamic simulation it may vary with time, while for steady state simulation, it is the unknown whose values are to be determined. A parameter, on the other hand, is known and does not vary with time in dynamic simulation. The parameters can become independent variables in an optimisation problem. All variables and parameter types are given unique integer code numbers in DASP starting from 1. For example temperature is 51, pressure is 50 and so on. Values are given in Appendix B8

6.2.2 MODULE OR MODEL ROUTINE

A module or model routine is a Fortran 77 subroutine, which contains the model equations of a whole or part of a unit operation, written in DASP module format which is made up of sections including initial, function evaluation, Jacobian evaluation, event processing, sparse Jacobian matrix setup and output sections. Some of these sections are optional. Details of these are described in Appendix B5.

6.2.3 UNIT MODULE

A unit module is defined as a lumped-parameter model of a whole or part of a unit operation, in which not more than one example of each variable type as described in

Appendix B5 appears, excluding input streams. To be able to handle an equilibrium stage as a unit module, the variable code numbering distinguishes between vapor and liquid type variables. For example the jacketed tank shown in Figure 6.1a, is made up of two unit modules as shown in Figure 6.1b.

6.2.4 STAGED MODULE

This is a module which is made up of a collection of identical stages as in a distillation or absorber column. Each stage has the same number of variables and parameters but may have a different number of input and output streams.

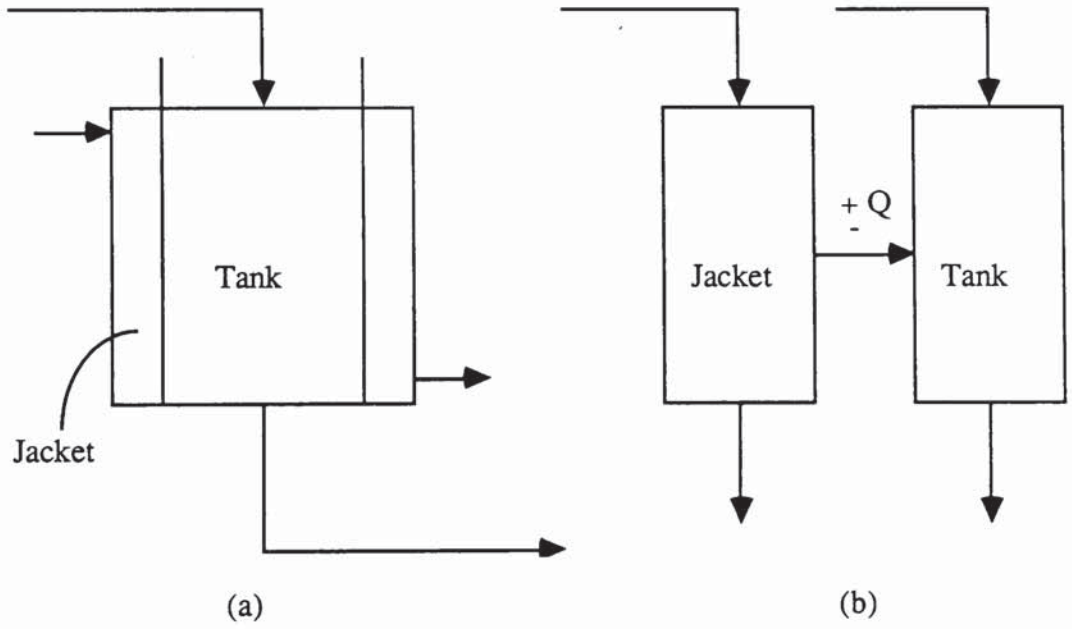


Fig 6.1 An example of translation of flowsheet to block diagram.

6.2.5 SECTIONALISED MODULE

A sectionalized module is made up of a number of sections, which are identical and each may have the same number of variables. However, the same parameters are applicable to all of them. For example, the modelling of distributed systems generates partial differential equations which can be solved by the method of lines. This results in sections which have identical structure and may have the same number of variables but the same parameters are applicable to each. In the same way, a stream splitter can also

be regarded as a sectionalized module in which each output stream is a section.

6.2.6 UNIT

A unit is the part of the user's problem described by a single module. It describes the distinct block of the transformed flowsheet which can be modelled by the module. For example, a splitter is one unit while a jacketed vessel is made up of two units. Units are also used to describe sources or sinks of material. All feed streams are regarded as originating from a source unit and all product streams as terminating at a sink unit.

6.2.7 STREAM

A stream is a connection between two units. This can be in the form of control lines, material flow or heat flow. It can also mean parameter or variable flow, in which the value of a parameter or variable is passed to or from a unit. Each of the stream types have a unique code number, given in Appendix B7

6.3 THE MODEL ROUTINE

A model routine defines the equations to be solved for the whole or part of a unit operation. The design of a model routine was arranged to fulfill a number of functions, including describing the equations to be solved and the calculation of the Jacobian matrix of the subsystem. The structure makes it possible to:

1. Assemble the equations to be solved very easily from the modules.
2. Write a new model routine without difficulty.
3. Communicate variables and parameters between modules.
4. Have a common structure of the model routine for all module types.
5. Have a structure such that modules are fairly independent of one another.

There conflicts between these requirements. However, these have been resolved in the structure which is of the form:

```
SUBROUTINE MODULE (IC, JS, T, X, DX, ICX, F, IEP, REP,
ICEP, CJ, IFLAG, RP, IP, CR, ICR)
INTEGER IC, JS, ICX (*), IEP (*), ICEP (*), IFLAG, IP, ICR (*)
REAL T, X(*), DX (*), F (*), REP (*), CJ, RP CR (*)
COMMON MODWK1/IU, LDNIN, LDNOUT, MVAR, MFUNC, MPMI,
MPMR, IPHSE, IUNC, NC, ID (20), ICVR (50), IPM (20)
COMMON MODWK2/FX (100), RPM (20), YF (20), XF (20), HV (20),
HL(20)
|
|       Descriptions of the purpose of the module, its
|       variables and parameters.
|
IFLAG = 0
IF (JS.EQ. 0) THEN
       — INITIALIZATION SECTION
ELSEIF (JS. EQ. 1) THEN
       — FUNCTION EVALUATION SECTION
ELSEIF (JS. EQ. 2) THEN
       — JACOBIAN EVALUATION SECTION
ELSEIF (JS. EQ. 3) THEN
       — OUTPUT SECTION
ELSEIF (JS. EQ. 4) THEN
       — EVENT CODE SECTION —
ELSEIF (JS. EQ. 5) THEN
       — TERMINAL SECTION —
ELSEIF (JS. EQ. -1) THEN
       — SPARSE JACOBIAN MATRIX ELEMENT SECTION
ENDIF
RETURN
END
```

In the sections that follow, the arguments in the common blocks and the argument list of the subroutine are described. Afterwards each of the above sections is described in turn.

6.3.1 DESCRIPTION OF COMMON BLOCK ARGUMENTS

IU the index of this unit in the vector of unit numbers, IUNOS(*).

LDNIN the logical device number for input of data.

LDNOUT the logical device number for output of data

MVAR the number of variables in this unit

MFUNC the number of equations in this unit

MPMI the number of integer type parameters

MPMR the number of real type parameters

IUNC on input it contains the unit number of this unit. Subsequently, it is used to

communicate information to this unit, regarding the status of adjacent units. When the RTRV routine is called to retrieve variable or parameter values from an adjacent unit, if IUNC is less than zero then the adjacent unit has been removed from the flowsheet.

NC the number of components in this unit

ID (*) the identity of the components in this unit

ICVR (*) integer work array

IPM (*) integer work array

FX (*) work array, used to communicate the values of variables/parameters between units when RTRV is called. Thus a variable with code number 51 has its value in FX (51).

RPM (*),YF (*),XF (*),HV (*),HL (*) are work arrays

6.3.2 DESCRIPTION OF ARGUMENT LIST OF THE MODULE

All have values on entry unless noted.

IC the code number of this module

JS section identifier. It indicates which section of the module is to be executed

T the current time

X(*) the current values of the variables of this module

DX (*) the current values of the derivatives of the variables of this module

ICX (*) the code numbers of variables of this module

F(*) real array that forwards the residuals of variable after function evaluation (JS=1) or the elements of the Jacobian matrix after Jacobian evaluation (JS=2). Calculated in the routine.

IEP (*) the integer type parameter values

REP (*) the real type parameter values

ICEP (*) the code numbers of the real type parameters

CJ a real parameter used in calculation of elements of Jacobian matrix.

IFLAG performance flag, set in the routine to a negative value if any fatal errors occur within the module, otherwise it is zero.

RP real type variable which can be used to communicate information between this module and the calling routine or used by this routine for other purposes. It is not modified by the calling program.

IP integer type variable. It has the same function as RP above.

CR (*) DASP real work array; must not be modified. It is equivalent to CORE(*) defined in the main program.

ICR (*) DASP integer work array; must not be modified. It is equivalent to ICORE(*) defined in the main program.

6.3.3 INITIALIZATION SECTION

In this section, the model routine is instructed which type of model equations are to be solved. It selects the options needed, which automatically determine the number of equations and variables. The options are set by the integer parameters, which are input by the user. The real parameters, the variable initial values and their derivatives are supplied by the user. The number and type of parameters of both integer and real types, and of variables are different for each module. These are described in Appendix B6.

For example, a SENSOR module, which can be described by either a zero, first or second order differential equation, has the following initialization setup.

1. The integer type parameters are as follows:

(i) MOPTN model option control parameter

= 0 zero order system (simple gain)

= 1 first order system

= 2 second order system

(ii) ICODEI code number of input variable

(iii) ICODEO code number of output variable

(iv) IDERIV derivative input option

= 0 no derivatives available

= 1 derivatives of variables provided.

2. The real type parameters depend on the model option chosen. However, in every case the following data are needed:

(i) ZI zero of input signal/variable

(ii) RI range of input signal/variable

(iii) ZO zero of output signal/variable

(iv) RO range of output signal/variable

(v) X0 zero offset of input signal/variable

(vi) Y0 bias of output signal

(vii) GAIN sensor gain

If MOPTN is 1 or 2 then

(viii) TAU time constant.

If MOPTN is 2 then

(ix) DAMP damping ratio.

3. The variable of the unit

(i) Y sensor output signal/variable

4. The derivative of the unit

(i) DY derivatives of the sensor output.

From these data, the following parameters are set:

a. Integer Type Parameters

The integer type parameters for this unit,are given by their code numbers as:

MPMI = 4

ICVR (1) = 1

ICVR (2) = 3

ICVR (3) = 4

ICVR (4) = 5

The parameters are read from a file or the terminal via LDNIN channel number by calling the UNIPMI routine as in

CALL UNIPMI (LDNIN, IEP, ICVR, MPMI, IFLAG)

b. Real Type Parameters

The number and type of the real type parameters are set according to the model option chosen as follows:

MPMR = 7 for zero order (MOPTN = 0)

= 8 for 1st order (MOPTN = 1)

= 9 for 2nd order (MOPTN = 2)

Then the code numbers of the selected real type parameters are set in ICEP (*) array and their values are read into REP (*) using:

CALL UNIPMR (LDNIN, REP, ICEP, MPMR, IFLAG)

c. Variables and Derivatives

The number of variables, equations and the code numbers of the variables are set as:

MVAR = 1

MFUNC = 1

ICX (1) = code of variable

Then the variables (and derivatives if IDERIV = 1) are read via:

CALL UNIVAR (LDNIN, IDERIV, X, DX, ICX, MVAR, IFLAG)

Note that in each case after a return from the relevant input routine, the values of the variables or parameters read in are checked to make sure that they fall within certain bounds. For example, MOPTN should be integer and either 0, 1 or 2 only. If any errors are detected, a negative value of IFLAG is set and control is returned to the calling

subprogram, which calls the error reporting routine.

6.3.4 FUNCTION EVALUATION SECTION

This is the section that calculates the residuals of the equations, given the values of the variables, parameters, derivatives and time. The input variables to this unit and any forcing functions are retrieved from their units of origin using the RTRV routine as follows:

CALL RTRV (ISNO, IFLAG, CR, ICR)

where

ISNO is the stream index of the input unit as given by the user in the connection matrix (see section 6.4). On return from the RTRV routine, it is checked that:

1. IUNC is not less than zero; otherwise it means that the respective unit has been disconnected from the flowsheet. In this case the values from this unit are not used in the calculation.
2. IFLAG is not less than zero. Otherwise, an error occurred during the retrieval of the input unit values and hence the calculation is abandoned and control returned to the calling subprogram.

The residuals of the equations are then calculated and control passed to the calling subprogram.

6.3.5 JACOBIAN EVALUATION SECTION

This is optional. If this is available, then the procedure is very much the same as in function evaluation except that instead of returning the residuals in F (*), the elements of the Jacobian matrix are calculated and returned in row order in F (*).

6.3.6 NONZEROS OF SPARSE MATRIX SETUP SECTION

This is also optional. However, when provided, the indexes of the nonzeros of the Jacobian submatrix are set up in row order via SETJAC routine as:

CALL SETJAC (IEQ, ISNO, MVR, IFLAG, IND, CR, ICR)

where

IEQ is the equation number in the unit

ISNO is the index of the stream (as in connection matrix)

MVR is number of variables concerned

IFLAG is the performance flag

IND is the indicator flag, with values as follows:

= 0 for nonzeros of this unit in the particular equation being processed

> 0 for nonzeros of other units

Thus for one equation, a first call transfers nonzeros of the unit, then subsequent calls are made for nonzeros of other connected units one after the other.

6.3.7 OUTPUT SECTION

This section, which is optional, can be used to output values of the variables and parameters of the module during an output time using LDNOUT as channel number.

6.3.8. EVENT CODE SECTION

Here the particular event code can be implemented, which will be executed when this routine is called with JS = 4. This is optional.

6.3.9 TERMINAL SECTION

The user can implement those statements which need to be executed at the end of the simulation in this section. This is optional.

6.4 TRANSFER OF VARIABLES BETWEEN MODULES

When a module is called, only the variables, their derivatives and the parameters of the module are transferred to the module. All the input stream variables are not immediately available to the unit. The RTRV routine is used to retrieve the values of variables, derivatives and parameters of other connected units or streams. It uses the units connection matrix MP (IU, *) of the process block diagram and the following information:

(a)	LOCVAR (IU)	index of the first location in CORE(*) of the variables of the unit with index IU
(b)	LOCDX (IU)	index of the first location in CORE(*) of derivatives of the unit with index IU
(c)	LOCPMI (IU)	index of the first location in ICORE(*) of integer type parameters of the unit with index IU
(d)	LOCPMR (IU)	index of the first location in CORE(*) of real type parameters of the unit with index IU
(e)	LOCCDE (IU)	index of the first location in ICORE(*) of code numbers of variables of the unit with index IU
(f)	LOCPMC (IU)	index of the first location in ICORE (*) of code numbers of real type parameters of the unit with index IU
(g)	NVARU (IU)	the number of variables of the unit with index IU
(h)	NEPMR (IU)	number of real type parameters of the unit with index IU

In the above, IU is the index of the unit whose variables, derivatives and parameters are to be retrieved.

Every module in DASP has a connection block, which determines in the module the order in which the connected streams are to be called to retrieve the values of the variables, derivatives and parameters needed in the module. A typical connection block

is shown in figure 6.2.

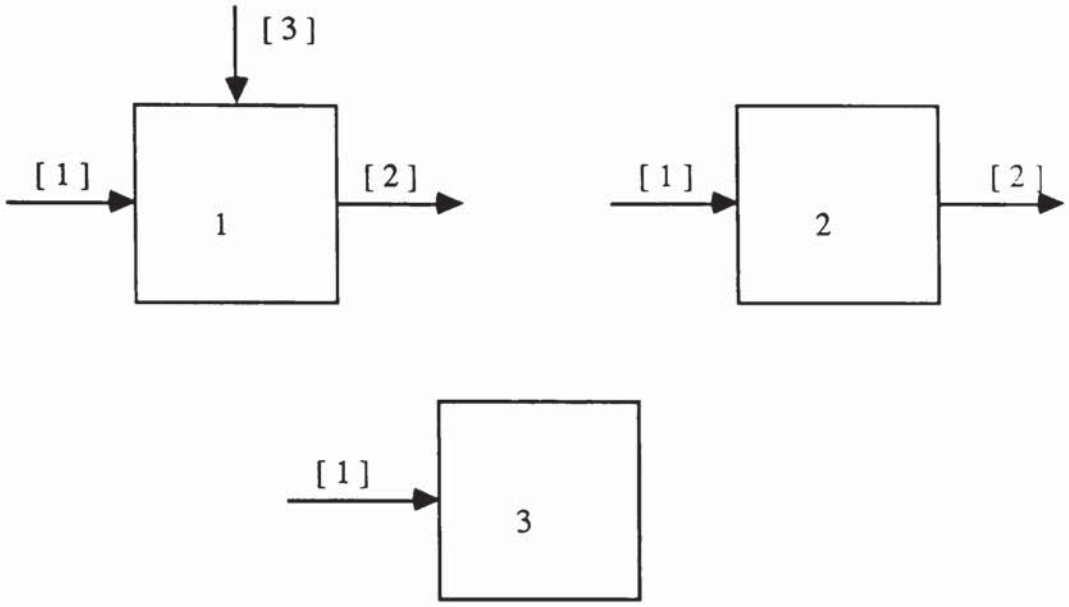


Figure 6.2 - Connection Blocks of Modules

In these blocks the numbers given inside the blocks are the unit numbers of these blocks while the numbers beside the streams connected to the blocks are the connection numbers of the blocks. These numbers indicate the order in which the unit numbers of the units connected to these blocks must be given in the array, MP(IU,*), which is the connection matrix. For example, let us connect the blocks in Figure 6.2 as shown in Figure 6.3.

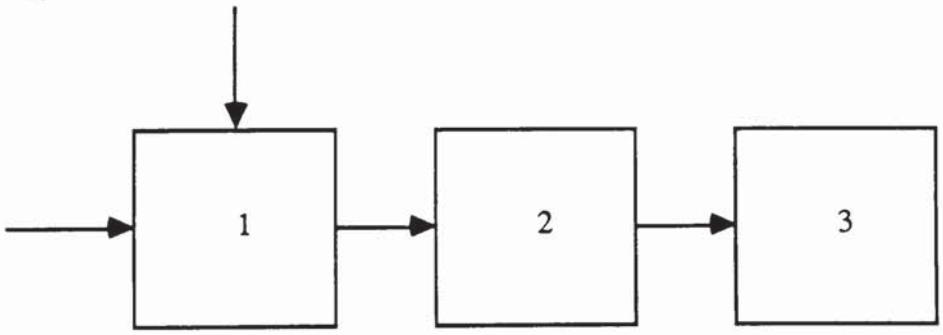


Figure 6.3 - Block diagram of a process

The connection matrix of unit number 2 are the unit numbers of those units connected to unit 2, given in the order shown in the connection block of unit 2. In the present case, the connection matrix of unit 2 are 1 , 3. Thus when unit 2 is called to calculate the

residuals of the modelling equations, the values of variables of unit number 1 will be retrieved by calling the RTRV routine with ISNO=1. The RTRV routine uses the index of unit number 1 to locate the storage positions of its variables, derivatives and parameters in CORE(*) and these values are returned to the calling unit in FX(*) and DFX(*) stored according to the value of the code numbers of the variables or parameters. Thus a variable with code number 51 is stored in position 51 in FX(*). The same procedure is repeated for any other units whose variable values are required in unit number 2.

6.5 ASSEMBLING OF EQUATIONS FROM MODULES

The strategy used in assembling the equations to be solved from modules is based on the fact that only the residuals of the equations are required during function evaluation, and no derivative evaluation is needed as in the sequential modular method or some equation solving methods which do not solve differential-algebraic systems. The principle in the equation-oriented approach is that the model equations should be written in the form:

$$\mathbf{r}_i = \mathbf{F}_i\,(t, \mathbf{y}, \mathbf{y}') \qquad . 6.4$$

$$\text{for } i = 1, N$$

where

$\mathbf{r}_i$ is the residual of equation i

$\mathbf{y}$ is a vector of the variables of the whole system

$\mathbf{y}'$ is a vector of the derivatives of all the variables.

In a modularly-organized equation-oriented approach, subsets of the above equations are coded in modules as in:

$$\mathbf{r}_j^{(k)} = \mathbf{F}_j^{(k)}\,(t, \mathbf{y}, \mathbf{y}') \qquad . 6.5$$

for $k = 1, Nu$

$j = 1, Mu^k$

where

Nu is the total number of modules

Mu^k is the total number of equations in module k

Thus

$$\mathbf{r} = (\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^{Nu})$$
$$\mathbf{F} = (\mathbf{F}^1, \mathbf{F}^2, ..., \mathbf{F}^{Nu}) \qquad .. 6.6$$
$$N = (Mu^1, Mu^2, ..., Mu^{Nu})$$

Since a global Newton-method is used, which does not require equations to be written in any order, the sequence of calling the modules to assemble these equations is immaterial. During function evaluations, the integrator forwards the values of the current time, TIME, the variables and derivatives XVAR(*) and DXVAR(*) respectively by calling GETFUN as:

```
CALL GETFUN(USRSUB,TIME,XVAR,DXVAR,FUNC,IFLAG,CORE,ICORE)
```

where

FUNC(*) will contain the residuals calculated

IFLAG should be set to a negative value if unable to compute residuals.

In GETFUN, using the pointers described in Section 6.4, the values of the variables, derivatives for the unit are located in XVAR(*) and DXVAR(*) vectors. Also the parameters, number of variables and parameters, number of components and their identities for the module are located in CORE and ICORE arrays as well as from the common blocks. The storage space of the residuals of the equations in the module within FUNC(*) vector is also located. All these data are forwarded to the module via the common block, MODWK1 as well as the augment list of the GETSUB subroutine .

which actually calls the relevant module by name. In the module, all the variables, derivatives and parameters of the connected input and output units, which appear in the equations of this module are retrieved from their respective positions in CORE(*) vector (actually XVAR(*) and DXVAR(*) vectors) as explained in Section 6.3.4. Then the residuals of the module equations are calculated and placed in their correct positions in the FUNC(*) vector. This is illustrated schematically in Figure 6.4.

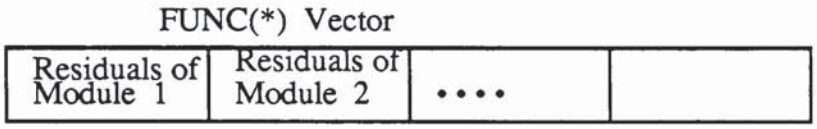


Figure 6.4 - Arrangement of residuals of modules in FUNC(*)

6.6 THE USRSUB OPTION

The USRSUB option offers an alternative to using the modules in DASP. It is equivalent to the fully equation-oriented approach where the problem to be solved is described by a large system of differential-algebraic equations. The subroutine, USRSUB, which is just a variable name, is an interface routine which either contains the equations to be solved or calls another subroutine which contains these equations. In all cases, the equations are formulated so that the residuals of each equation are returned instead of the derivatives.

The USRSUB routine is structured, in much the same way as the module routine described in Section 6.3, except that since all the equations and variables are contained within the same "module", there is no need to retrieve the values of variables and derivatives of connected modules or units. The structure and functions of a USRSUB routine are described in Appendix B4.

The main advantages of the USRSUB option are:

1 The user can describe his problem using this option, especially if it is a small problem or if the model is not available in the DASP module library.

2 It is simple to write and all the features available in DASP can also be used even in this case.

3 The user can modify the model as he wishes.

6.7 CONCLUSION

Two approaches to generation of the model equations to be solved in an equation-oriented environment have been described, namely the DASP module option (modularly organized equation-oriented approach) and the USRSUB option (fully equation-oriented approach). The structure of the DASP module, which has functional divisions of initial, function evaluation, Jacobian evaluation sections, etc, uses this powerful feature of the CSSL to simplify the complexity of equation-oriented simulator modules. The main limitation of the present design is that in a module, there is an arbitrary division of variables and parameters, although several choices are offered via the use of integer parameters or options control variables. This is unnecessary and a new approach has been described in Appendix C, which allows the user to choose any of the variables of the module as the unknown.

CHAPTER 7

THE EQUATION SOLVING PACKAGE

7.1 INTRODUCTION

The core of any dynamic simulation package is the integrator which solves the dynamic model equations to produce the time varying trajectories of the variables in the model. In Chapter 2, it was argued that simultaneous solution of DAEs not only gives more accurate results than the sequential modular approach but also provides advantages in the structuring of the model. DASSL (Petzold, 1983) has been found to be a robust DAE solver and can handle stiff systems effectively. This was confirmed by running the test problems provided, by Enright et al (1975).

DASSL is a general-purpose integrator and in order to use it to handle the special requirements of a process dynamic simulator such as DASP, several changes were necessary not only to adapt and enhance its features to the requirements of DASP, but also to reduce storage requirements and provide better error diagnostics. Also, in order to demonstrate the potential of using the same model for both steady state and dynamic simulations without any modifications the Newton-Raphson method and a modified Broyden's method were implemented to solve the nonlinear algebraic equations of the steady state option.

First, a brief description of DASSL is given in Section 7.2, while Section 7.3 presents the changes made to it. In Section 7.4 a brief description of the features of the non-linear equation solvers, NRSUB and BROYDN is given. Section 7.5 describes the service routines needed for solving the linear equations generated by DASSL, NRSUB and BROYDN and the routines for the calculation of the residuls and the Jacobian matrix of the model equations.

7.2 DESCRIPTION OF DASSL

DASSL (Petzold, 1983), acronym for Differential-Algebraic System SoLver, can solve equations of the type:

$$\mathbf{F}\ (\mathbf{y}',\ \mathbf{y},\ t) = \mathbf{0} \qquad \ldots\ 7.1$$

given $\mathbf{y}\ (t_0) = \mathbf{y}_0$ and $\mathbf{y}'\ (t_0) = \mathbf{y}'_0$. In the above equation, $\mathbf{F}$, $\mathbf{y}$ and $\mathbf{y}'$ are N dimensional vectors, where N is the number of equations. It is useful for solving two general classes of problems which cannot be handled by standard ODE solvers. For the first class, it is not possible to solve for $\mathbf{y}'$ explicitly to rewrite (7.1) above in the standard form (equation 3.2). For the second class it is possible in theory to solve for $\mathbf{y}'$, but it is impractical to do so. For example, to convert $\mathbf{A}\mathbf{y}' = \mathbf{B}\mathbf{y}$ to standard form, we must multiply by $\mathbf{A}^{-1}$. If $\mathbf{A}$ is a sparse matrix, $\mathbf{A}^{-1}$ may not be sparse. So it is advantageous to solve the equations in their original form to maintain the structure of the system, as advantage can be taken of the structure to reduce memory requirements.

DASSL uses the k^{th} order BDF to approximate the derivative, where k ranges from one to five. On every step, it chooses the order, k, and step size, h, based on the behaviour of the solution. Newton's method converges most rapidly when the initial guess, $\mathbf{y}_{n0}$ is accurate. DASSL obtains an initial guess for $\mathbf{y}_n$ by evaluating the polynomial which interpolates the computed solution at the last k+1 times, t_{n-1}, t_{n-2}, t_{n-3} ..., $t_{n-(k+1)}$ at the current time, t_n. An initial guess for $\mathbf{y}'_n$ is obtained by evaluating the derivative of this polynomial at t_n. Once $\mathbf{y}_{n0}$ is found, Newton's method is used to solve for $\mathbf{y}_n$ as in equation (3.20), except that in general the derivative is approximated by the k^{th} order BDF instead of the backward difference of $\mathbf{y}_n$. If we rewrite equation 3.19 as

$$\mathbf{F}\ (\alpha\, \mathbf{y}_n + \mathbf{b}_n,\ \mathbf{y}_n,\ t_n) = 0 \qquad \ldots..\ 7.2$$

where α is a constant which changes whenever the stepsize or order changes, and **b** is a vector which depends on the solution at past times, the solution of this equation by Netwon's method gives:

$$y_n^{m+1} = y_n^m - C * G^{-1} * F(\alpha' y_n^m + b_n, y_n^m, t_n) \quad \ldots\ldots 7.3$$

The iteration matrix $G = \left[\frac{\partial F}{\partial y'_n} + \alpha \frac{\partial F}{\partial y_n}\right]$ is computed and factorised, and is then used for as many time steps as possible. In general, if the value of α when **G** was last computed is different from α', then equation (7.3) may not converge. The constant C in equation (7.3) is chosen to speed-up the convergence when $\alpha \neq \alpha'$, and is given by:

$$C = \frac{2}{(1+ \alpha'/\alpha)} \quad \ldots\ldots 7.4$$

The rate of convergence ρ of equation (7.3) is estimated when two or more iterations have been taken by:

$$\rho = \frac{\| y_n^{m+1} - y_n^m \|^{1/m}}{\| y_n^1 - y_n^0 \|} \quad \ldots 7.4a$$

The iteration has converged when

$$\frac{\rho}{1 - \rho} \| y_n^{m+1} - y_n^m \| < 0.3 \quad \ldots\ldots 7.5$$

If $\rho > 0.9$ or $m > 4$ and the iteration has not yet converged, certain rules are used to decide whether to reduce the stepsize or calculate an iteration matrix or both based on the current approximations to y_n, y'_n and α_n. The step is then attempted again.

The linear system generated is solved using the LINPACK Library subroutine package (Dongarra et al (1979)), which can either be dense or have a banded Jacobian matrix strucuture. The iteration matrix is computed by finite differences, although there is an option for the user to supply the analytical Jacobian. After the corrector has converged, an error test is made to determine whether the solution satisfies a local error tolerance specified by the user. This test is satisfied whenever $C \| y_n - y_{n0} \| \leq 1$, where C is a constant which depends on the order and recent stepsize history of the method. The constant C is chosen to control both the variable stepsize local truncation error, and the error in interpolated values of y_n between mesh points. If the error test is satisfied, the code takes another step, otherwise either the stepsize or the order or both are reduced and the step repeated.

To determine the stepsize and the order for the next step, the code estimates what the error would have been if the last few steps had been taken at constant step size at the current order k and at k-2, k-1 and k+1 (Shampine and Gordon, 1975). If these estimates increase as k increases, the order is lowered. If they decrease, it is raised. The new stepsize h_{n+1} is chosen so that the error estimate based on taking constant stepsize h_{n+1} at order k_{n+1} satisfies the error test.

If the initial values of the derivatives are not given, DASSL uses the backward Euler method (Carnahan and Wilkes, 1980) in conjunction with a damped Newton iteration to compute the initial derivatives This method does not always work. The code is arranged so that a driver routine, called SDASSL, allocates storage based on the information flags

stored in INFO(*), checks for illegal input and other errors and, if no errors are encountered, it calls the one-step solver SDASTP to advance the step over each time step. Communication between SDASSL and the other routines is via one common block, which contains pointers to the work assays, and argument lists. DASSL calls two routines which must be supplied by the user, RES to compute the function values and JAC to calculate the Jacobian matrix for user-supplied analytical Jacobian option. The parameter, IDID gives the status of the solution process. A negative value indicates error, which is usually written via an error routine. A positive value suggests successful completion.

7.3 ADAPTATION OF DASSL

A number of modifications were made to the original code (described in Section 7.2) and some new features added to enhance its performance and capabilities. Some of these changes are described below.

7.3.1 MODIFICATION OF ARGUMENT LISTS OF SUBROUTINES

The argument lists of the subroutines were rearranged and modified to add new control options and new arrays which provide some special facilities. These include addition of:

1. KDAE, which indicates whether the error test should be performed on algebraic components of the variable vector or not. This is an option in DASP.
2. EDYN(*), an array, whose i^{th} component is zero if the i^{th} variable is algebraic, otherwise it is 1. This array is ignored if no distinction is to be made between the algebraic and differential variables.
3. NZ, the total number of non-zeros in the sparse Jacobian matrix .
4. IEQN(*), an array containing the equation numbers of non-zeros of sparse Jacobian matrix.
5. JVAR(*), an array containing the variable numbers of non-zeros of sparse Jacobian matrix.

The work arrays, RWORK(*) and IWORK(*), used by DASSL, were modified to include work space for the sparse option. The arrays, RPAR(*) and IPAR(*), which were dummy arrays, are now equivalent to CORE(*) and ICORE(*) work arrays of DASP. The argument, JAC which provides the analytical Jacobian matrix, if available, was removed from argument list. This is because DASP assumes that the analytical Jacobian is provided via the same routine that provides the numerical Jacobian (see section 7.5.2). The INFO(*) array, which provides information to DASSL about the options chosen by the user was modified to accommodate the new options included.

7.3.2 REPLACEMENT OF THE ERROR ROUTINE, XERRWV

This error routine was replaced by the DASP error routine, REPORT which provides better diagnostics of the error and writes further information from an error messages file. Also code numbers were allocated to the error types that may occur in DASSL as is the case in DASP. See section 5.6 for further details of error reporting.

7.3.3 THE INCORPORATION OF THE SPARSE MATRIX OPTION

DASSL can only handle systems whose Jacobian matrix has a dense or banded structure. Since most large chemical processes have a sparse Jacobian matrix structure, the INFO (*) vector was modified so that INFO (6) = 2 means the use of the sparse option. This meant modification and recoding everywhere the Jacobian matrix is used in the calculation or where pointers for the matrix are set. The sparse linear system solver, MA28 (Duff 1977) was installed and interfaced with DASSL.

7.3.4 IMPLEMENTATION OF EVENT PROCESSING OPTION

Event processing is an integral part of the DASP package. To detect a state event, the integrator must be supplied with all the relevant information (see Chapter 8 for details). The event processing package was interfaced to DASSL via two routines, EVTEST and TIMCAL. After every prediction phase in the integration process, EVTEST is called to test if the event flag has been set and if so whether any threshold values have been crossed. In the case when a threshold value has been crossed, the routine TIMCAL is

called to calculate the exact time of its occurance, TOUTEV. This new time becomes the output time of integration, ie. TOUT = TOUTEV. A flag EVFLAG is set to .TRUE. to indicate that the threshold value has been crossed.

7.3.5 CHANGE OF EXECUTION LOGIC OF THE INTEGRATOR

In the normal integration process in DASSL, after a successful step, the integrator calculates the step size and order to use for the next step, based on considerations mentioned in Section 7.2. However, it may be necessary during simulation for a user to continue the integration using constant step size or to restart from a previous time after pertubation. To accommodate these changes efficiently, the calculation of the step size and order for the next step is deferred until the beginning of the step. Then INFO (7) is tested for any such changes mentioned above. If no such changes were made then the new step size and order are calculated as normal, otherwise, the integration is continued using the constant step or restarted.

7.3.6 NEW APPROACH TO INITIALIZATION OF DAES

In Chapter 3, it was discussed that certain classes of DAEs are not solvable by available ODE codes which have been adapted for solving DAEs. For the class of solvable DAEs, consistent initial conditions must be provided which must satisfy the equations to be solved at the initial time. Thus the DAE system

$$\frac{dy_1}{dt} = f_1\,(t, y_1, y_2, z) \qquad \text{.....7.6a}$$

$$\frac{dy_2}{dt} = f_2\,(t, y_1, y_2, z) \qquad \text{.....7.6b}$$

$$0 = g\,(t, y_1\, y_2, z) \qquad \text{.....7.6c}$$

consists of 3 equations and 5 variables namely y_1, y_2, dy_1/dt, dy_2/dt and z. This

means that only 2 (= 5-3) variables can be chosen arbitrarily and the rest must be determined from the equations. The implicit formulation of equations used in the design of DASP in which more than one derivative term can occur in an equation means that we cannot associate one derivative term with any one equation as in the case of equations written in standard form. Also the equation may be nonlinear in z, the algebraic variable and so z may not be calculated explicitly from the algebraic equation.

DASP evolves a strategy whereby it is assumed that the user provides the correct values of all the differential variables, y_1 and y_2 and an estimate of the algebraic variables, z. Thus the variables in the system now become dy_1/dt, dy_2/dt and z, which are to be solved using a nonlinear equation solver. Estimates of the derivatives of the differential variables are obtained by equating the derivatives to zero before function evaluation and calculating the function values using the values of y_1, y_2 and z. The calculated function values become the estimates of the derivatives.

The Newton-Raphson method is used to solve for the new variables. In order to calculate the correct Jacobian matrix, the routine that sets up the matrix was modified to cater for this case of DAE initialization, since in this case the variables are the derivatives and the algebraic variables and not the differential and algebraic variables. This strategy have been found to work well and provides the correct initial conditions in 1 to 5 iterations. This is because the equations are at most mildly nonlinear in z but always linear in the derivatives. The choice of the differential variables as the known while the derivatives and the algebraic variables are the unknown is for convenience. In theory, any N number of the set of differential, algebraic and the derivatives can be chosen as the unknown variables, where N is the number of equations.

7.4 THE NONLINEAR EQUATION SOLVERS

In this section, the implementation of the Newton-Raphson (NR) and the Broyden

algorithms are described. These are used to solve the nonlinear algebraic equations of the steady state simulation in DASP. The same linear equation solvers used with DASSL are also used with these two routines.

7.4.1 THE NEWTON-RAPHSON ROUTINE, NRSUB

This routine solves a system of nonlinear algebraic equations by the Newton-Raphson method, which consists of the repeated linearization of the equations using the latest variable values and the solution of the resulting systems of linear equation using a suitable linear equation solver. The Jacobian matrix generated at the m^{th} iteration can be of full, banded or sparse structure. The Newton-Raphson method is represented by

$$\mathbf{J}_m \Delta \mathbf{y}_m = -\mathbf{F}(\mathbf{y}_m) \qquad \text{..... 7.7}$$

where

$\Delta \mathbf{y}_m = \mathbf{y}_{m+1} - \mathbf{y}_m$

$\mathbf{F}(\mathbf{y}_m)$ = residuals of the equations

m is the iteration counter.

After $\Delta \mathbf{y}_m$ is calculated from equation (7.7) the next iterate values are calculated from

$$\mathbf{y}_{m+1} = \mathbf{y}_m + d_m \Delta \mathbf{y}_m \qquad \text{..... 7.8}$$

The value of d_m in the above equation is usually 1.0 for the Newton-Raphson method. However, as explained in Chapter 3, the Newton-Raphson method may diverge if the initial estimates are far away from the solution. In order to reduce the occurrance of this, the step taken is limited by requiring that $\mathbf{y}_{m+1}$ satisfy the following inequality:

$$\left[\sum_{i=1}^{N} \mathbf{F}_i^2 (\mathbf{y}_{m+1})\right]^{1/2} < \left[\sum_{i=1}^{N} \mathbf{F}_i^2 (\mathbf{y}_m)\right]^{1/2} \qquad \text{..... 7.9}$$

If y_{m+1} does not satisfy the above inequality, a fraction, d_m of the predicted step, Δy_m is used in the Newton-Raphson equation. d_m is calculated as follows:

$$d_m = \frac{1}{1 + \eta_{2/} \eta_1} \qquad \text{..... 7.10}$$

where

$$\eta_2 = \left[\sum_{i=1}^{N} F_i{}^2 (y_m + \Delta y_m) \right]^{1/2} \qquad \text{..... 7.11}$$

$$\eta_1 = \left[\sum_{i=1}^{N} F_i{}^2 (y_m) \right]^{1/2} \qquad \text{.....7.12}$$

This procedure clearly restricts the size of the step taken and thus reduces the chances of divergence. However, if the inequality (equation 7.9) is satisfied, then d_m is taken as 1, thus using the full Newton-Raphson step.

During the first call of this routine, with JSTART = 0, a series of initialisation processes are carried out, including allocation of storage spaces to work arrays, setting of flags used internally within the program and checking of the consistency of input values. If any errors are detected, the performance flag, IFLAG, is set to the negative of the code number of the error type and control is passed to the calling program, ASSUB. All function evaluations are carried out via the interface routine, GETFUN, which assembles the model equations from the modules or from the user-supplied routine, USRSUB, and all the Jacobian calculations are done via GETJAC, which calculates the Jacobian matrix. The latter is not inverted, but the LU-factors of the matrix (Finlayson, 1980) are used. The Jacobian matrix is usually scaled using the reciprocal of the maximum entry in each row. This entry is also used to scale the residuals of the equations.

The convergence error test is done using a weighted square root norm (Burden et al,

1981) which is given by:

$$\| (y_{m+1} - y_m)/W_m \|_2 < \varepsilon \qquad \ldots\ldots 7.13$$

where

W is the vector of maximum values of the variables so far,

ε is the user-supplied relative error tolerance

$\|.\|_2$ is the weighted square root norm.

The iteration is stopped if equation 7.13 is satisfied or if the maximum number of iterations has been exceeded without convergence, in which case a suitable error flag is set before control is passed to the calling program.

7.4.2 THE BROYDEN ROUTINE, BROYDN

BROYDN uses a modified Broyden's method (Crowe, 1984; Gallun and Holland, 1980) to solve a system of non-linear algebraic equations. It exploits and preserves the sparsity of the system to be solved and does not require matrix factorization at each step of the iterative procedure. This saves computation time in large systems. In the Gallun and Holland modification of the original Broyden's method, the Jacobian matrix is factored only once and afterwards updated as the iteration proceeds. However, Lucia (1983) pointed out that this procedure is best carried out with periodic Jacobian restart, in which the Jacobian matrix is re-evaluated after a specified number of iterations. In BROYDN, the Jacobian matrix is re-evaluated after 10 iterations.

As in the case of the Newton-Raphson method, the structure of the Jacobian can be full or sparse and the algorithm is solved using a suitable linear equation solver. The Jacobian and function evaluation is carried out as explained in Section 7.4.1 and the next iterate is calculated using equation (7.8), with y_{m+1} required to satisfy equation (7.9) except that d_m is calculated as follows (Gallun and Holland, 1980):

$$d_m = ((1 + 6\eta)^{1/2} - 1) / 3\eta \qquad \ldots\ldots 7.14$$

where

$$\eta = \eta_3/\eta_1 \qquad \text{..... 7.15}$$

$$\eta_3 = \left[\sum_{i=1}^{N} F_i{}^2 (y_m + d_m \Delta y_m)\right]^{1/2} \qquad \text{..... 7.16}$$

First d_m is set to 1.0 and if the inequality is not satisfied, then d_m is calculated using equation (7.14). If the norm is not satisfied after 5 trials through the use of equations 7.7 through 7.9 and equations (7.14) through equation 7.16 then the Jacobian is re-evaluated using the latest variable values. One particular advantage of the Broyden's routine is the option of the use of other forms of initial Jacobian, especially if the initial values are not available. In this case, either the identity matrix or an option to calculate diagonal matrix entries instead of the full matrix may be used. This may result in considerable saving in Jacobian matrix computation. All other calculations use the same principles described in Section 7.4.1.

7.5 THE SERVICE ROUTINES

A number of service routines are used by DASSL, NRSUB and BROYDN to assist in solving the user's problem. These include the function evaluation routine (GETFUN), the Jacobian evaluation routine (GETJAC) and the linear algebraic equation solving routines (the dense matrix solving LINPACK library routines (Dongarra et al, 1979) and the sparse matrix solving routine, MA28 (Duff 1977)).

7.5.1 FUNCTION EVALUATION ROUTINE, GETFUN

GETFUN interfaces to the model routines via GETSUB or to the users model equations via USRSUB interface routines. Its function is to calculate the residuals of the equations written in the form:

$$\mathbf{r} = \mathbf{F}\ (\mathbf{y}', \mathbf{y}, \mathbf{p}, \mathbf{t}) \qquad \text{..... 7.17}$$

where

r is a vector of the residuals of the equations.

During function evaluation, DASSL, NRSUB or BROYDN calls GETFUN with the values of the current time, TIME, variables, XVAR(*) and their derivatives, DXVAR(*) and GETFUN using the pointers to the positions of the parameters, variables and derivatives of the modules, transfers these values to the model routines for calculation of the residuals. FUNC(*) vector returns the calculated residuals to the calling subroutines with the residuals, stored module by module.

7.5.2 JACOBIAN EVALUATION ROUTINE, GETJAC

All Jacobian matrix evaluations are done via GETJAC routine. It uses the value of JFBS to determine the structure of the Jacobian matrix. JFBS has values of 0 for full or dense matrix, 1 for banded matrix and 2 for the sparse Jacobian matrix. In general, the Jacobian matrix of a chemical process can be represented as in Figure 7.1.

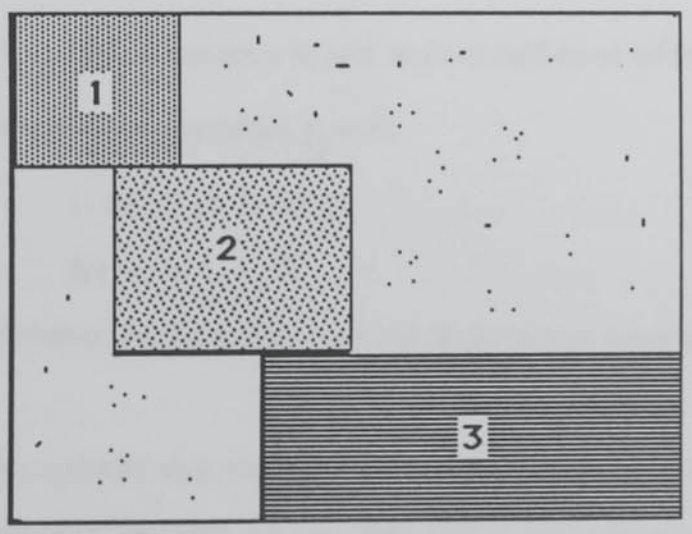


Figure 7.1 Structure of a Jacobian matrix

In the above figure, each of the shaded blocks represents the Jacobian elements of a subsystem, which in this case, is a module, while the scattered dots are the nonzeros of the input and output units to the modules. This structure is only one of the many

possible representations. If the Jacobian matrix is dense, then an NxN matrix results and the elements are calculated and stored in the PD(*) vector, column by column. For a banded matrix, using the band widths supplied by the user, the Jacobian matrix is calculated and stored in PD vector as:

$$PD(irow, j) = \left[\frac{\partial F_i}{\partial y_j} + C\frac{\partial F_i}{\partial y'_j}\right] \quad \ldots\ldots 7.18$$

where

irow = i - j + ML + MU + 1

i is the i^{th} equation, i = 1, N

j is the j^{th} variable, j = 1, N

ML is the lower band width of Jacobian matrix, excluding the diagonal

MU is the upper bandwidth of the Jacobian matrix, excluding the diagonal

C is a parameter supplied by the integrator.

Note that an equation is said to have half-band widths (lower and upper) if equation i involves only variables y_j with

$$i - ML \leq j \leq i + MU \quad \ldots\ldots 7.19$$

for j, i = 1, ..., N

However, because PD(*) is a vector the actual Jacobian is stored column by column.

For a sparse system, most of the elements of the Jacobian matrix are zero, usually the non-zero elements are less than 10% of the total elements. It is required that the user supply the non-zero entries by giving their equation and variable numbers in the vectors IEQN(*) and JVAR(*) respectively. If the equations are generated from modules, then DASP calculates the equation and variable numbers by one or more calls to the model routines with JS = -1 as explained in Appendix B5. The Jacobian matrix is thus stored in the PD(*) vector row by row, in the order in which they appear in IEQN(*) and JVAR(*) vectors.

The Jacobian matrix can be calculated using the finite difference method or the user can supply the Jacobian of the system in the Jacobian evaluation sections of the model routines or USRSUB.

7.5.3 LINEAR EQUATION SOLVERS

The linear algebraic equations generated after the linearization of the non-linear equations using either the BDF-Newton-Raphson in DASSL, Newton-Raphson in NRSUB or Broyden's method in BROYDN are all solved by direct methods using a suitable linear equation solver depending on the structure of the Jacobian matrix. For the full matrix, direct LU-decomposition with partial pivoting (Burden et al, 1981) is used. For the banded option, the LINPACK Library sub-routine package (Dongarra et al, 1979) are used. To handle the sparse option, the sparse matrix package MA28 (Duff, 1977), was installed and interfaced to DASP. DASSL was modified to accommodate the sparse option as explained in Section 7.3. The NRSUB and BROYDN were designed to handle the full and sparse matrix structures efficiently.

7.6 CONCLUSION

In this Chapter, the implementation of the numerical methods and algorithms needed for solving the differential-algebraic equations that arise in dynamic simulation have been described. The modified DASSL integrator can handle both small and large DAEs by using the sparse option for the large systems. These routines have been implemented in DASP as the equation solving package.

CHAPTER 8

THE EVENT PROCESSING PACKAGE

8.1 INTRODUCTION

Events frequently occur in chemical processing, especially during start-up and shut-down of chemical plants, investigation of a plant's response to disturbances and batch operations. Generally, an event defines a point in time beyond which the state of the present system is undefined. If the time of occurrence of an event is known a priori, then such an event is called a time event. For example, an event can be specified by a user-defined forcing function or the reset of a state variable value. On the other hand, a state event is one in which the time of its occurrence is defined implicitly through the attainment of a desired state, for example, a concentration of some chemical species, a temperature or a tank level. Process dynamic simulators generally accommodate time events (e.g. DYNSYL (Patterson and Rozsa, 1980)) while discrete event simulators are designed to handle state events (eg. GASP IV (Pritsker and Hurst, 1973). However, a few other simulators specifically designed to handle batch operations can accommodate both time and state events (e.g. BOSS (Joglekar and Reklaitis, 1984). In order for DASP to study the start-up and shut-down of chemical processes, it must be able to handle both types of events.

Event processing is analogous to integration over discontinuities as described in Section 3.4, where the methods for handling such discontinuities were given. In order to accommodate an implicitly defined event (i.e. state event), the simulator must first identify that such an event has occurred, then it must calculate the time of its occurrence, and return control to the executive which then calls the event logic appropriate for that event. The integrator is then called again to continue from where it stopped. This disrupts the smooth integration process, especially with multi-step methods of the BDF-type. These methods are based on polynomial interpolation over an interval of

several steps, and they break down when the interval includes a discontinuity in a derivative of degree lower than that of the polynomial being used. Hence a restart is necessary. In this section, the way DASP handles state and time event processing and restarting after the event logic has been processed are described. The structure of the event processing package is shown schematically in Fig. 8.1. Also described is the MODIFY routine which offers several features needed in a dynamic simulator. A discussion at the end summarizes the advantages and limitations of the approach used.

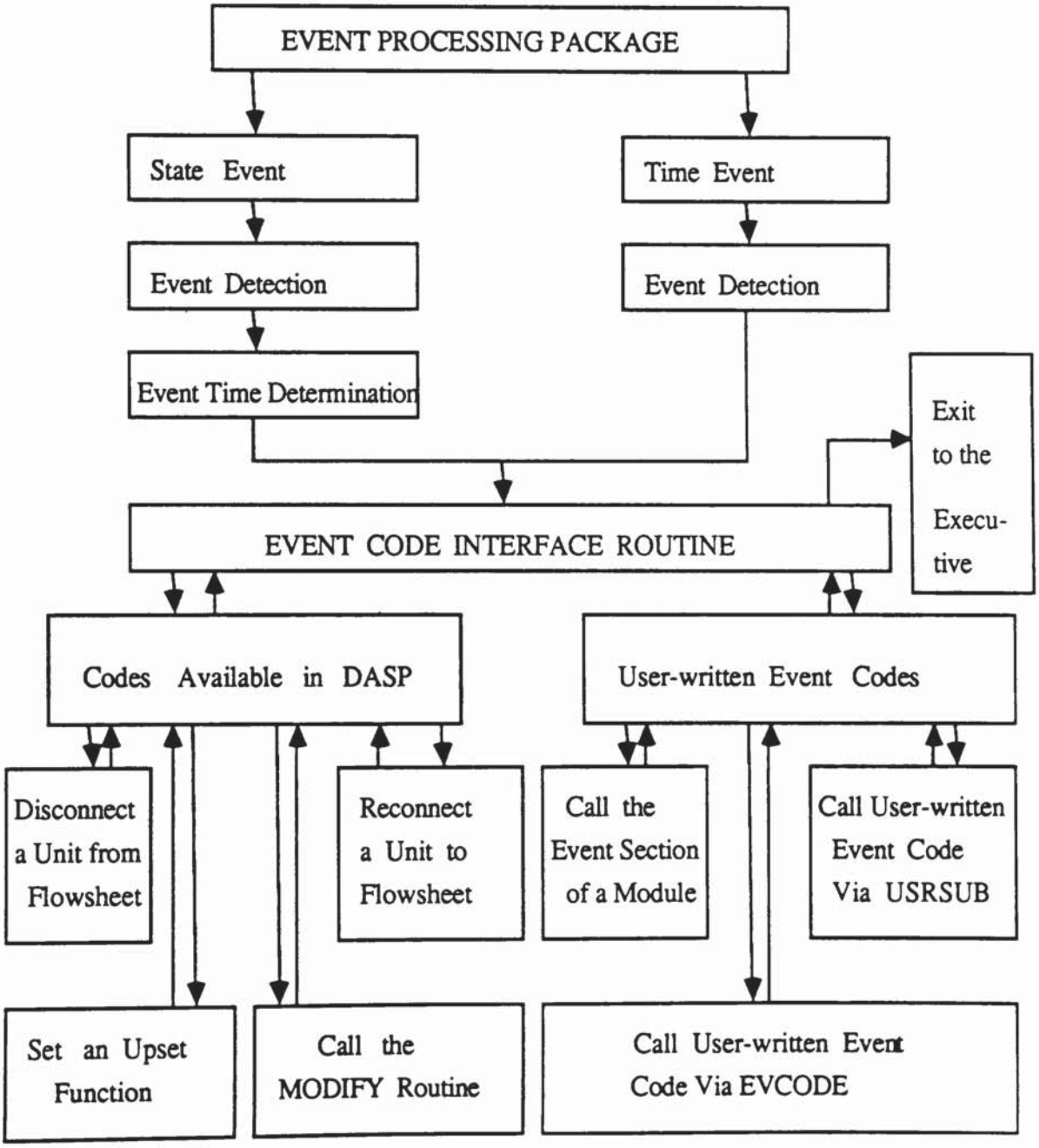


Fig. 8.1 Structure of the Event Processing Package

8.2 EVENT DETECTION

DASP can handle state events which occur in a predetermined sequence during a simulation. This means that multiple events which can occur in any sequence are not accommodated, although this limitation could be removed by making a few modifications to the event detecting code, as explained later. A state event, which is given the code number 2 in KECODE (INDEV), is uniquely determined by the following state event information:

1. The unit number of the state variable stored in KEUNIT(INDEV) whose threshold value will be used to determine the event time.
2. The stage/section number of state variable (for staged/sectionalised modules) stored in KSUNIT(INDEV).
3. The code number of the variable stored in KVARN0(INDEV)
4. The threshold value stored in TRSVAL (INDEV)
5. The threshold tolerance stored in TRSTOL (INDEV)
6. The direction of threshold crossing stored in IDCROS (INDEV)

where INDEV is the index of the state event in the sequence. Items 1 to 3 identify the variable in the system for the case of units available in the library. However if the user supplies his own model via USRSUB routine, then items 1 and 2 are not needed, and item 3 becomes the index of the state variable in the list of variables in XVAR(*) vector.

After every predictor evaluation during integration, the predicted value of the variable and the direction of threshold crossing are used to determine if the threshold value has been crossed by calling the routine, EVTEST. If this has happened, then EVFLAG is set to TRUE and the coefficients of the variable are used to determine the state event time in a Newton's iteration scheme. This is explained in the next section

If events are allowed to occur in any sequence, then for state events, it will be necessary to check all possible events and identify which one, if any, has occured, instead of

checking for the occurence of the next event in the sequence as is the case at the moment. This procedure may require an event calender to be kept, which shows which events have taken place and when, and which ones are yet to take place. Thus the scanning of the possible events will be on those that are yet to occur. Alternatively, events could be allowed to occur in a predetermined sequence but in this case one of a set of possible events will occur at the event time. For example, an event sequence consists of two events, namely EVENT No 1 and EVENT No 2. EVENT No 2 could be described as follows:

> "Simulate the process until either the concentration of component A has reached 40 mol % or 15 minutes has elapsed, whichever comes first".

This defines two possible events, of which only one would be allowed to happen. This type of event could be implemented as proposed in Appendix C.

8.3 EVENT TIME DETERMINATION

DASSL maintains the coefficients of a q^{th} order polynominal for each variable, in an Nx(q + 1) array , where q is the order of integration (Petzold, 1983). The coefficients are those of a scaled divided difference representation of the polynomial. This polynominal is used to determine the state event time. This procedure is similar to that used in BOSS (Joglekar and Reklaitis, 1984). If t_{n-1} is the time when the last step was successfully calculated and t^{p}_{n} the time when the predicted values were evaluated, then the root of the function

$$p(t) = \varphi(t) - V \qquad ..8.1$$

where

p (t) is the function whose root is to be determined

$\varphi(t)$ is the polynominal which determines the value of the variable

V is the threshold value of the variable in TRSVAL(INDEV)

determines the event time. Since the root is bounded between t_{n-1} and t^P_n this iteration is easily achieved. In DASP, the routines TIMCAL, EVROOT and DDPOLY determine the exact time ,TEV, of threshold crossing. When this time is determined, then this becomes the new output time, and a new step size is calculated to coincide with the new time. EVFLAG is set to TRUE to indicate that an event has taken place. DASSL then integrates from t_{n-1} to TEV and returns with the values of the variables and derivatives at the new output time, TEV.

A time event, is uniquely defined by the following information:

1. Event index, INDEV,which gives the serial number of the next event in a sequence.
2. Event time, TEV, calculated as TIME + EVTIME(INDEV)
3. Time event code, stored in KECODE(INDEV).

If the next event to occur in the sequence is a time event, then the integrator interface, DERIV makes sure that the next output time, TOUT, is not greater than the time of its occurrence, TEV. If TOUT is greater than TEV, then TOUT is set to TEV and EVFLAG is set to TRUE. This will indicate to the Executive program that an event has taken place when the integrator returns with the values of the variables at the new output time, TEV.

8.4 THE EVENT LOGIC

At an event time, control is passed to the event processing interface routine, STESUB routine, which displays the message:

*** EVENT PROCESSING INTERFACE ROUTINE ***

*** EVENT MENU ***

0 = Exit.

1 = Change variable/parameter values in a unit.

2 = Disconnect a unit from the flowsheet.

3 = Reconnect a unit to the flowsheet.

4 = Call a unit module event section.

5 = Call user written event code via "EVCODE" routine.

6 = Call user written event code via "USRSUB" routine.

The user is prompted for the index of an option. These options are explained below.

8.4.1 EXIT

This option must be entered in order to exit from STESUB routine. When chosen it checks if the topology of the system to be solved has changed, in which case the incidence matrix of the flowsheet is checked for correctness and consistency. If no errors were detected, then the positions of the variables, their code numbers and derivatives and error tolerances are rearranged so that those units which are still connected are serially arranged. This is necessary so that all the variables of the connected units will be serially arranged in the variable vector, XVAR(*) as required by the integrator. If the sparse numerical option was being used to solve the problem, new equation and variable numbers are calculated. Also, if the final time of integration has been reached, a new final time is requested. JSTART is also set to zero, to indicate that the integration is to be restarted. On the other hand, if the flowsheet has not been changed and the values of the variables have not been changed, then the integration is to be continued from where it was stopped. In this case, control is simply returned to the Executive routine, DASPM, which calls the integrator again.

8.4.2 CHANGE VARIABLE/PARAMETER VALUES

In choosing this option, the user intends to change the value of either variable or

parameters. EVSUB1 is the routine that accomplishes this task. The following message is displayed:

*** THE OPTIONS AVAILABLE ARE ***

0 = Exit.

1 = Change the value of a variable.

2 = Change the value of a parameter.

and the user is prompted for the index of an option. To change the value of a variable in a unit the following information is needed:

1. The unit number of the variable
2. The stage/section number if the unit is a staged or sectionalised module.

For a parameter change, only the unit number is required. With this information, all the names of the variables or parameters within the chosen module are displayed with their indices. In the case of a user-supplied model routine via USRSUB, only the serial number of the variable as given by the user in the XVAR(*) array during initialization will be needed. In all cases, the current value of the chosen variable or parameter will be displayed before the user enters a new value for the variable or parameter. For a variable change, a flag, IEVENT is set to a positive value to indicate to the Executive that the integration needs to be restarted. More variable or parameter changes can be made if necessary or a return to STESUB is made by choosing the EXIT option above.

8.4.3 DISCONNECT A UNIT FROM THE FLOWSHEET

This option is intended to assist in simulating the start-up or shut-down of a chemical plant, as well as batch processes. The original topology of the plant can be changed during the simulation, and some parts of the flowsheet can be simulated independent of the other parts, provided those parts were part of the original topology. This is achieved by disconnecting those units not needed at the present time and then simulating the new

flowsheet. As an illustration, suppose it is intended to simulate the flowsheet shown in Figure 1.1 of Chapter 1, using the event logic of Section 1.4. After event No 1, valve V2 and V3 and tank T2 are not needed. They are thus disconnected from the flowsheet and valve V1 and tank T1 now form the new flowsheet to be simulated. Similarly, after event No 2, valve V1 and V3 and tank T2 are not needed. The new flowsheet is now valve V2 (which should now be reconnected) and tank T1. At event No 3, valves V1 and V2 should be disconnected while valve V3 and tank T2 should be reconnected. If event No 3 is modified for continuous operation, then at event No 3, all the units which were previously diconnected can now be reconnected from the flowsheet. In these procedures, the disconnected units are dynamically inactive, because the values of their state variables do not change with time. As many units as necessary can be disconnected from the flowsheet by repeated use of this option, a unit at a time. In each case the following message is displayed:

** ENTER A UNIT NUMBER **

The unit whose number is entered, is disconnected by setting KUC (IU) = - IUN

where

IUN is the unit number of the unit as given by user in IUNOS(*) vector.

KUC(*) is an integer vector which holds the connection flags of the units. Initially, it contains the unit numbers of the units. When a unit is removed from the flowsheet, its unit number in KUC(IU) is set to the negative of the unit number to indicate that the unit is disconnected. Reconnection means reverting to the positive unit number.

IU is the index of the unit in IUNOS(*) vector, which contains the unit numbers.

8.4.4. RECONNECT A UNIT TO THE FLOWSHEET

This option is the direct opposite of the option described in Section 8.4.3. A unit which has been previously disconnected from the flowsheet can be reconnected via this option.

In this case, the connection flag is set to the positive number, i.e. KUC (IU) = IUN. It is also possible to cut a stream from one unit and connect it to another without disconnecting the unit from the flowsheet. This is most useful during start-up and shutdown of processes as well as in batch operations. However a stream can only be connected to a unit at a point where there was a connection before, called a port. The following data are needed to connect one unit to another:

1 The unit number IUN of the unit to reconnect to the flowsheet.

2 The stream number of the stream previously connected to unit IUN.

3 The unit number, IUC, of the unit to which unit IUN is to be connected.

4 The stream number of the stream previously connected to unit IUC.

With these four data, the flowsheet of the plant is modified to create the new flowsheet. The unit number of the unit to be reconnected is reset to its absolute value, which signifies that the unit is now connected. This option is illustrated using the block diagram in Figure 8.2

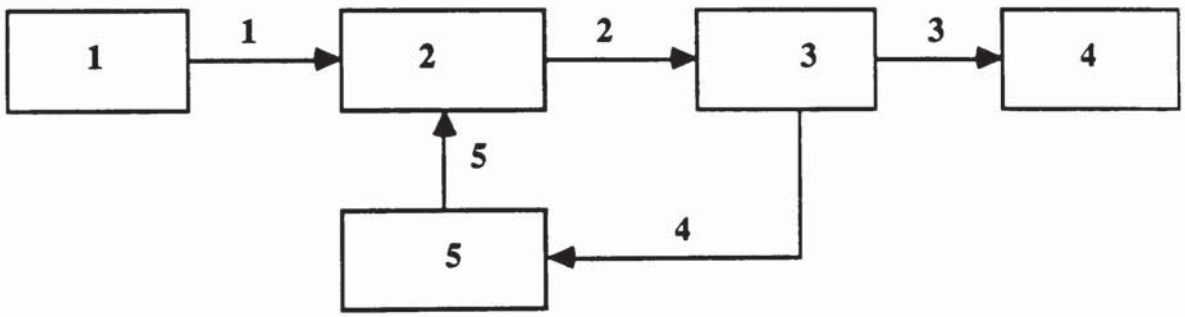


Fig. 8.2 A Block Diagram of a Flowsheet

The numbers in the boxes are the unit numbers as given by the user, while the streams have their stream numbers beside them. The incidence matrix of this block diagram is given by the information in Table 8.1. It is also assumed that the connection blocks of the modules (unit 2, 3 and 5) and those of input and output streams (unit 1 and 5 respectively) are as given in Figure 8.3.

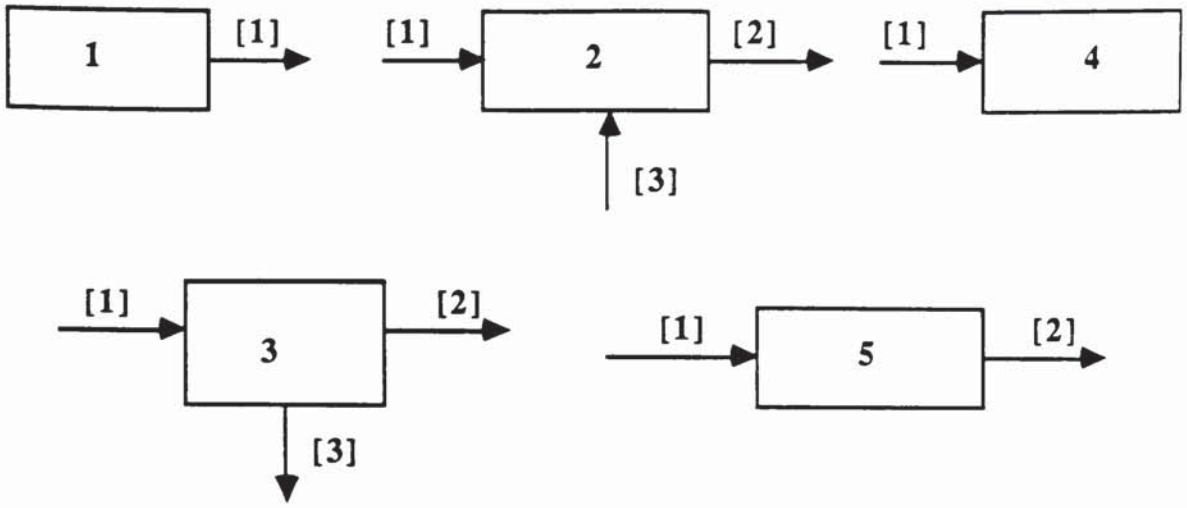


Fig. 8.3 Connection Blocks of Units in Fig. 8.2

Table 8.1 - Incidence Matrix of Figure 8.2

Stream Nos, ISNOS (*)	1	2	3	4	5
Units of origin, IUFROM (*)	1	2	3	3	5
Units of termination, IUTO(*)	2	3	4	5	2

Note the connection block gives information about the order of naming the units connected to the unit being described. From the connection blocks and the block diagram, the connection matrix can be constructed as presented in Table 8.2.

Table 8.2 - Connection Matrix of Figure 8.2

Unit, IU	1	2	3	4	5
No of units connected to unit, IU, NS	1	3	3	1	2
Units connected to unit, IU given in the MP (IU, *) order of the connection block	1	1 3 5	2 4 5	3 3 2	

Supposing that units 3 and 4 have previously been disconnected from the block diagram of Figure 8.2 and it is required now to reconnect them. This will be done by reconnecting stream No 2 to unit No 2, stream No 4 to unit No 5 and stream No 3 to unit No 3. For reconnecting stream No 2, the following values are needed:

Unit to reconnect	3
Previous stream connected to this unit	2
Unit to reconnect unit 3 to	2
Stream previously connected to unit 2	2

The subroutine, EVSUB3 checks the validity of these data against the incidence matrix of Table 8.1. If they are not valid, as for example, if the user gives a unit no of a unit not in the block diagram, these errors are displayed and the user may start again. In the current case, the index of the unit previously connected to unit number 3 is determined from the incidence matrix for unit number 3 and the new unit to be connected to it is placed in this position. The same is done for unit number 2. The index of unit no 2 (the unit previously connected to unit 3) is determined from the connection matrix, MP (3,*). That is, using the example above, unit 2 was in position 1 of unit 3 and the number of the new unit, (in this case unit 2) is put in this position. The same is done for the case of the connection matrix of unit no 2. This procedure is then repeated for stream numbers 3 and 4. Although the procedure described above may seem tedious, it works and ensures that the stream is connected to the specified unit. This approach could be incorporated into a graphical front-end program.

8.4.5 CALL A MODULE EVENT SECTION

A module either from the DASP Library or added by the user in DASP module format can have suitable event logic included as indicated in Section 6.3 of Chapter 6. At present all the modules in the DASP Library have no event code in this section. If this option is chosen, the unit number of the module is requested and the event section of this module is called to execute the event code.

8.4.6 CALL USER-WRITTEN EVENT CODE VIA EVCODE

A user can write an event code via the interface routine EVCODE as:

```
SUBROUTINE EVCODE (USRSUB, IFLAG, IEVENT, CORE, ICORE)
EXTERNAL USRSUB
```

```
      INTEGER IFLAG, IEVENT, ICORE(*)
      DOUBLE PRECISION CORE(*)
            |
            |   } Event code
            |
      RETURN
      END
```

where

USRSUB is the dummy name for a user supplied routine,

IFLAG is a performance flag, set to a negative value if an error condition occurs, otherwise it returns with a zero value,

IEVENT is an integer, which is set to a positive value if the topology of the system has been changed, otherwise it is zero. Possible changes include:

1. Change of the number of variables and equations.
2. Addition or removal of model routines by disconnecting or reconnecting a unit from or to the flowsheet.

CORE(*) is a real storage space

ICORE(*) is an integer storage space

This subroutine should be compiled and loaded with DASP Library. It must be emphasized that in order to write this routine, the user must know the functions of the various common blocks in DASP. An example use of this option is illustrated in Chapter 9. When this option is chosen, this subroutine is called to execute the event code.

8.4.7 CALL A USER-WRITTEN EVENT CODE VIA "USRSUB"

This option is only applicable to the case where a user supplies his model equations using "USRSUB" interface routine. In this routine, the Section with JS=4 is reserved for the event code. When this option is selected, the USRSUB routine is called with JS=4 to execute the event code. An example use of this option is illustrated in Chapter 9.

8.5 THE UPSET FUNCTIONS

With the help of the upset functions, the user can perturb the system to study the effect of certain changes in the values of a parameter and forcing function on the variables of the system being simulated. The following upset functions are available:

1. Step function
2. Ramp function
3. Pulse function

The step function is described by the following:

$$X = \begin{cases} x0step \text{ if } t < t0step \\ x1step \text{ if } t \geq t0step \end{cases}$$

where

X is the value of the parameter at time, t

t0step is the time of step upset

x0step is the value of parameter before the step upset

x1step is the value of parameter after the step upset

A ramp function is defined as:

$$X = \begin{cases} x0ramp \text{ if } t < t0ramp \\ x0ramp * (1.0 + m * (t - t0ramp)) \end{cases}$$

where

t0ramp is the time of ramp upset

x0ramp is the value of parameter before the ramp upset

m is the slope of the ramp function

A pulse function is defined by:

$$X = \begin{cases} x0plse \text{ if } t < t0plse \\ x1plse \text{ if } t0plse \leq t < t1plse \\ x0plse \text{ if } t > t1plse \end{cases}$$

where

t0plse is the initial time of pulse upset

t1plse is the final time of pulse upset

x0plse value of parameter before pulse upset

x1plse value of parameter during pulse upset.

Any of the functions above can be chosen, causing a parameter or forcing function value to vary according to the function throughout the period specified for the function by the user. This can be used to simulate the effect of fluctuations in the values of feed stream variables on the values of the variables of interest in the system. New functions, such as sine functions and random number generators, can easily be included. The necessary data are input by the user. An example of the use of the upset function is described in Chapter 9.

8.6 THE MODIFY ROUTINE

It is of fundamental importance for a user to have access to all the variables, parameters and some control information in an interactive environment during a simulation session in a dynamic simulation package. This facility is provided in DASP by the MODIFY routine. It provides several facilities and options which can be combined to assist in the simulation of a chemical plant. In general, a user can use the MODIFY routine to do the following:

1. Modify any of the variables and parameters in the model, as well as the control variables of the package. These include the integration parameters, the error tolerances, event processing information and so on.

2. View any of the variables, parameters or control information mentioned above.

3. Plot the variables specified by the plotting information.

4. Set an upset function against a parameter or forcing function, so that the effect of the change in the value of the parameter or forcing function on the variables of the system can be studied. The step, ramp and pulse functions are available.

The MODIFY routine is by default called in the initial region after data input from files, but in the terminal region it is one of the terminal options which can be selected by the user. It is also possible to have access to this routine in the Dynamic region by setting a break flag so that the MODIFY routine will be called after a user-specified number of steps or interactions. It is also called if certain errors occurs which can be corrected using this routine. When called, the following message is displayed:

DASP MODIFY ROUTINE

OPTIONS AVAILABLE ARE

---

0 = EXIT	1 = OPTIONS	2 = FILES
3 = OUTPUT	4 = PLOTTING	5 = EVENT
6 = INTEGRATION	7 = EQUATIONS	8 = VARIABLES
9 = TOLERANCE	10 = INTERUPT	11 = UPSET
12 = DISPLAY	13 = GRAPH	

ENTER AN OPTION NUMBER

These options are explained below. However, it should be mentioned that any of these options can be assessed independently by referencing its number as IND in the argument list. For example, a call of MODIFY as

accesses only the OPTIONS while a call with IND = 0 provides access to all the options. The detailed meaning and values of the variables described here are given in the glossary, Appendix B7.

1 EXIT

This option must be chosen to exit from this routine. Whenever this routine is called and the user does not want to use its facilities, this option should be entered to continue the simulation. However, if this routine is called with IND not equal to zero, then control is returned to the calling sub-program after execution of the specific option.

2 OPTIONS

With this option, the user can modify the values of the control options, which include the following:

1 Simulation mode option, MODE

2 Source of model routine option, KMODEL

3 Type of error tolerance option, KTOL

4 Non-negativity option, NONNEG

5 Model discontinuity option, IDISCN

6 Mono/multicomponent option, LCOMP

7 Output time option, LPRNT

8 Plot option, LPLOT

9 Event processing option, LEVENT

10 Integration parameters default option, MDEFLT

11 Jacobian matrix structure option, JFBS

3 DATA FILES

The user has access to all the data files names which are:

1 General input data file, CINDAT

2 Topology data file, CTOPOL

3 Output or result file, CRESLT

4 Initial values data file, CUNIT

5 DASP work file, CWORK

6 Components data file, COMDT

8.6.4 OUTPUT

The following information about the variables whose values are to be displayed can be modified:

1 Output level , INDEX

2 Terminal display flag, LTERM

3 Number of variables to display, NVPRNT

4 The unit numbers of the variables to display, LUPRNT (*)

5 The stage or section numbers of the variables to display, LSPRNT (*)

6 The code numbers (or serial numbers for user-supplied routine via USRSUB) of the variables to display, LVPRNT (*)

7 The names of the variables to display, CPRNT (*)

8 The units of the variables (for the case of user-supplied model routine), CUPRNT (*)

5 PLOTTING

Information about the variables to plot include:

1 The channel number to plot to, LDNPLT

2 Number of variables to plot, NVPLOT

3 The unit numbers of the variables to plot, LUPLOT (*)

4 The stage/section numbers of the variables to plot, LSPLOT (*)

5 The code numbers (or serial numbers) of the variables to plot, LVPLOT (*)

6 The names of variables to plot, CPLOT (*)

6 EVENT

The following data about event processing can be modified:

1 Number of events, NEVENT

2 Code numbers of the events, KECODE (*)

3 Event times, EVTIME (*)

4 Unit numbers of threshold variables, KEUNIT (*)

5 Stage/section numbers of threshold variables, KSUNIT (*)

6 Code numbers (or serial numbers) of threshold variables, KVARNO (*)

7 Values of threshold variables, TRSVAL (*)

8 Threshold tolerances, TRSTOL (*)

9 Direction of threshold crossings, IDCROS (*)

10 Names of events, CENAME (*)

7 INTEGRATION

The following integration parameters are available for modification:

1 Initial time of simulation, T0

2 Final time of simulation, TFIN

3 Integrator method flag, MFDYN

4 Differential-algebraic equation flag, KDAE

5 Minimum stepsize to use, HMIN

6 Initial stepsize to use, H0

7 Maximum stepsize to use, HMAX

8 Maximum order of integration, KORDX

8 EQUATION

This information is needed to describe the nature and structure of the non-linear equations and their Jacobian matrices both for the steady state and dynamic simulation. These include:

1 Maximum number of iterations to perform, KMAX

2 The non-linear equation solver method flag, MFALG

3 Jacobian matrix availability option, LJAC

4 Flag for initial Jacobian calculation, KJAC

5 Lower band of banded Jacobian matrix, ML

6 Upper band of banded Jacobian matrix, MU

7 Total number of non-zeros in sparse Jacobian matrix, NZ

8 Equation numbers of non-zeros in sparse Jacobian matrix, IEQN (*)

9 Variable numbers of non-zeros in sparse Jacobian matrix, JVAR (*)

9 VARIABLES

The option can be used to change the values of any variables and parameters in the system to be solved. The description of this option is given in section 8.4.2.

10 TOLERANCE

The following scalar or vector error tolerances can be modified:

1 Scalar relative error tolerance, TOL

2 Vector relative error tolerance, RTOL (*)

3 Vector absolute error tolerance, ATOL (*)

11 INTERRUPT

It is possible for the user to insert breakpoints so that after a number of steps or iterations, control is passed to the user to use the MODIFY routine or to interupt the simulation. Interrupt information include:

1 Interrupt or break flag, LINTRP

2 Total number of interruptions/breaks taken so far, INTNUM

3 Break point frequency, NINTRP.

12 UPSET

When the upset option is selected, the following message is displayed:

UPSET FUNCTION ROUTINE

If any of the upset functions has been set previously, ie. upset flag, KUPSET is 1, 2 or

3, then the message

"UPSET" FUNCTION FLAG IS ON

is displayed, where "UPSET" is substituted by either "STEP" or "RAMP" or "PULSE" depending on the value of KUPSET. Then the setting of the upset function are displayed. Otherwise the message

UPSET FUNCTION FLAG IS OFF

The upset options available are:

0 = EXIT

-1= SET UPSET FUNCTION OFF

1 = SET STEP FUNCTION ON

2 = SET RAMP FUNCTION ON

3 = SET PULSE FUNCTION ON

Whichever function (1-3 above) is selected, the following information will be needed:

1 The unit number of the parameter to upset,

2 The stage number of the parameter to upset (for staged module),

3 The code number or serial number of the parameter to upset.

These upset functions are described below.

i EXIT

Select to exit from this routine

ii SET UPSET FUNCTION OFF

Selecting this option, the user intends to cancel the upset function. In this case, the values of the parameters remain at their present values, and KUPSET is set to 0.

iii SET STEP FUNCTION ON

In addition to the information given previously, the following values are needed:

1 Time of step upset, t0step

2 Parameter value before upset time, x0step

3 Parameter value at step upset, x1step

iv SET RAMP FUNCTION ON

In addition to the information provided previously, the following values are needed:

1 Time of ramp upset, t0ramp

2 Parameter value before ramp upset, x0ramp

3 Value of slope of ramp function, x1ramp

v SET PULSE FUNCTION ON

In addition to the parameter identification information given above, the following data are also needed:

1 Initial time of pulse upset, t0plse

2 Final time of pulse upset, t1plse

3 Parameter value before pulse upset, x0plse

4 Parameter value during pulse upset, x1plse

Note that only one parameter can be upset at a time. However, this can be changed in the program by making the variables in common block /UPSET/ arrays instead of scalar.

13 DISPLAY

This facility can be used to view the values of variables, parameters and control information if selected. Usually, all the information in an option are displayed. The options available are the same as those described for the MODIFY routine.

8.7 DISCUSSION ON EVENT PROCESSING PACKAGE

This chapter has presented the implementation of the event processing package, which includes detection of an event, determination of the time of its occurrence, an interface to the event code to be executed after an event has taken place. Also included are certain features which may be combined to form an event code. This includes changing the values of variables and parameters, connecting and disconnecting units from the

flowsheet and the setting up of an upset function against a parameter or forcing function. However a user can write an event code either via the event section of a module or USRSUB routine or a seperate program using EVCODE interface routine.

The method used in implementing the modification of a flowsheet by disconnecting or reconnecting a unit or stream is not user-friendly enough. It is possible to make mistakes during the process. However it offers a way of removing dynamically inactive units as may be required during start-up and shut-down of a chemical process. It also reduces the dimension of the problem. What is needed is a good front-end to make the user interface more user-friendly.

The upset functions can be used to simulate how fluctuations in the values of feed stream variables or forcing functions affect the values of variables of interest in the system being simulated. This is most useful in checking the stability of a system and in checking the working of a chosen control scheme. However, this feature has a limited application as only one variable or parameter can be upset at any one time.

One of the most useful features in the event processing package is the MODIFY routine, which can be used to monitor the progress of the simulation dynamically, using a combination of plotting a selected number of variables and modifying or viewing the values of variables as the user may wish. Thus the event processing package offers the tools needed in dynamic simulation.

CHAPTER 9

DEMONSTRATION OF DASP

9.1 INTRODUCTION

In this chapter, example problems are used to demonstrate the features of DASP. The emphasis is on showing that the various options and features work and where possible the results are compared with literature values. The demonstration problems are divided into five sets, namely Example type I, II, III, IV and V. The first set deals with the solution of stiff systems of differential equations taken from the literature. Type II are test cases for the nonlinear equation solvers, NRSUB and BROYDN described in Chapter 7.

In Example type III, some DAE examples are solved using the USRSUB option. This shows how a user can supply model equations in a fully equation-oriented environment. To test the working of some of the modules in DASP, Example type IV problems are used. Finally, Example type V shows the working of certain features in DASP, including event processing, the rerun option, the use of upset functions and the modification of a flowsheet. A discussion of the examples and the results concludes this chapter.

9.2 Example Type I - Stiff Differential Equations

In this section, five stiff systems of differential equations are used to test the integrator, DASSL. In each case the equations describing the problem, the initial conditions and the initial and final times of integration are given. The detailed results are given in Appendix A.

9.2.1 DESCRIPTION OF EXAMPLE TYPE I PROBLEMS

1. Problem No 1 (Ref: Michelsen, 1976, probl 2)

This problem is a dynamic model for a fluid-bed reactor. In the equations given below, y_1 and y_3 represent temperature of the particle phase and the fluid phase, and y_2 and y_4 represent the partial pressure of the reactant in these phases. The problem is not only stiff but parameter sensitve. The analytical solution of $y_1(\infty)$ is 1211.

Model Equations:

$$y'_1 = 1.3\,(y_3 - y_1) + 1.04 \times 10^4\, k\, y_2$$

$$y'_2 = 1.88 \times 10^3\,(y_4 - y_2\,(1+k))$$

$$y'_3 = 1752 + 267y_1 - 269y_3$$

$$y'_4 = 0.1 + 320y_2 - 321y_4$$

$$k = 0.0006 \exp\,(20.7 - 15000/y_1)$$

Initial conditions:

$y_1\,(0) = 759.167$, $y_2\,(0) = 0.0$, $y_3\,(0) = 600$, $y_4\,(0) = 0.1$

Initial time: 0.0 , Final time: 1000.0

2 Problem No 2 (Ref: Chan et al, 1978, Probl no 1)

(Ozone Decomposition)

Model Equations:

$$y'_1 = -y_1 - y_1\, y_2 + a\, b\, y_2$$

$$a\, y'_2 = y_1 - y_1\, y_2 - a\, b\, y_2$$

$a = 1/98$, $b = 3.0$

Initial Conditions

$y_1\,(0) = 1$, $y_2\,(0) = 0$

Initial time = 0.0 , Final time = 50.0

3 Problem No 3 (Ref: Bui 1981, Problem 1)

This system exhibits an oscillatory behaviour, with a stiffness ratio as high as 3×10^4.

Model Equations:

$$y'_1 = 77.27\,(y_2 - y_1 y_2 + y_1 - 8.375 \times 10^{-6} y_1^2)$$

$$y'_2 = (-y_2 - y_1 y_2 + y_3)/77.27$$

$$y'_3 = 0.161\,(y_1 - y_3)$$

Initial conditions

$y_1(0) = 3 \quad y_2(0) = 1 \quad y_3(0) = 2$

Initial time = 0.0 , Final time = 300

4 Problem No 4 (Ref: Michelsen 1976, probl no 1)

In a closed system of three components, the following reaction path can occur:

$$A \xrightarrow{k_1} B$$

$$B + C \xrightarrow{k_2} A + C$$

$$2B \xrightarrow{k_3} C + B$$

Model Equations:

$$\frac{dC_A}{dt} = -k_1 C_A + k_2 C_B C_C$$

$$\frac{dC_B}{dt} = k_1 C_A - k_2 C_B C_C - k_3 C_B^2$$

$$\frac{dC_C}{dt} = k_3 C_B^2$$

Calculate the reaction pathway for $k_1 = 0.04$, $k_2 = 10^4$ and $k_3 = 3 \times 10^7$

Initial conditions:

$C_A(0) = 1$, $C_B(0) = 0$, $C_C(0) = 0$

Initial time = 0.0 , Final time = 10

5 Problem No 5 (Ref: Enright et al, 1975, problem C1)

Model Equations:

$$y'_1 = -y_1 + y_2^2 + y_3^2 + y_4^2$$

$$y'_2 = 10y_2 + 10(y_3^2 + y_4^2)$$

$$y'_3 = -40\,y_3 + 40\,y_4^2$$

$$y'_4 = -100y_4 + 2$$

Initial Conditions:

$y_1(0) = y_2(0) = y_3(0) = y_4(0) = 1$

Initial time = 0.0 , Final time = 20.0

9.2.2 SOLUTION OF EXAMPLE TYPE I

Example type I problems were solved using the USRSUB option. In each of the problems, a subroutine, USRSUB, was prepared, in which the equations were formulated as described in Chapter 6 and Appendix 4. This program was then compiled using the RM Fortran 77 compiler on the IBM PC AT and the compiled program linked with the DASP Fortran library as described in Appendix B3. Two data files were needed for each problem, one containing the general input data, while the other contains the values of all the variables. Each of the example problems was simulated from the initial time, T0 to the final time, TFIN. Table 9.1 shows the results of the simulation and the literature results where available. The full details of the results can be found in Appendix A.

Table 9.1 - Results of Example Type I Problems

(A = DASP results; B = Literature results)

Problem No		Time (hr)	y_1	y_2	y_3	y_4
1	A	1000.0	1.2107E3	1.2746E-4	1.2083E3	4.3859E-4
	B	∞	1.2110E3	-	-	-
2	A	50.0	4.2847E-4	1.3898E-2		
	B	50.0	4.28E-4	-		
3	A	300.0	4.8312	1.2584	3.1688	
	B	-	-	-	-	
4	A	10.0	8.4139E-1	1.6236E-5	1.5859E-1	
	B	10.0	84137E-1	1.6234E-5	1.5861E-1	
5	A	1.0	3.333E7	2.6456E+4	4.000E-4	2.00E-2
	B	-	-	-	-	-

These problems proved good enough test cases for the DASSL integrator. Problem No 5 proved the most difficult to solve and required the use of a very small steplength. However, as can be seen from the results of other problems for the cases where literature results are available, the computed solutions are highly accurate for the moderately low error tolerance of 10^{-3} used.

9.3 EXAMPLE TYPE II - NONLINEAR ALGEBRAIC EQUATIONS

The main objective in solving these test problems is to show use of the nonlinear algebraic equation solvers, NRSUB and BROYDN, described in Chapter 7, to solve nonlinear algebraic equations. The problem chosen for this purpose is one which has been shown by various workers to be a suitable test problem.

9.3.1 DESCRIPTION OF EXAMPLE TYPE II PROBLEMS

1. Problem No 6 (Ref: Shacham 1986, Problem 2)

This is a system of equations, which represents the rate of reaction of different steps in a chemical reaction. At the steady state solution, all the reaction rates must be equal to zero.

Model Equations:

$$r_1 = 1 - x_1 - k_1 x_1 x_6 + kr_1 x_4$$

$$r_2 = 1 - x_2 - k_2 x_2 x_6 + kr_2 x_5$$

$$r_3 = -x_3 + 2k_3 x_4 x_5$$

$$r_4 = k_1 x_1 x_6 - kr_1 x_4 - k_3 x_4 x_5$$

$$r_5 = 1.5 (k_2 x_2 x_6 - kr_2 x_5) - k_3 x_4 x_5$$

$$r_6 = 1 - x_4 - x_5 - x_6$$

where

k_1, kr_1, k_2, kr_2 and k_3 are specified reaction rate coefficients, x_1 to x_6 are quantities in moles of the different species present in the reaction and r_1 to r_6 are the reaction rates.

The parameter values are as follows:

$k_1 = 31.24$, $kr_1 = 2.062$, $k_2 = 0.273$, $kr_2 = 0.02$, $k_3 = 303.03$

Initial Estimates:

$x_1 = 0.99$, $x_2 = 0.05$, $x_3 = 0.05$, $x_4 = 0.99$, $x_5 = 0.05$, $x_6 = 0.0$

9.3.2 SOLUTION OF EXAMPLE TYPE II PROBLEM

This problem was solved using the initial guesses shown in Section 9.3.1 above. The iteration converged in 3 iterations for the Newton Raphson method and 7 iterations for the Broyden method.

9.4 EXAMPLE TYPE III - DAE SYSTEMS USING USRSUB OPTION

One of the two main ways of providing the equations to be solved in DASP is through the use of the USRSUB option. In this case the user writes a Fortran 77 subroutine. called USRSUB, in which the problem to be solved is described using the format explained in Appendix B4. All the problems solved here were coded in different USRSUB routines.

9.4.1 DESCRIPTION OF EXAMPLE TYPE III PROBLEMS

1 Problem No 7 (Ref: Holland & Liapis, 1983)

$$\frac{dy}{dt} = z - y$$

$$0 = -0.5\,y + z - 2$$

Initial Conditions:

$y(0) = 0.5$

$z(0) = 9/4$

$y'(0) = 7/4$

$z'(0) = 7/8$

Initial time = 0.0 , Final time = 0.5

This problem as it is, is overdefined. For two equations in which only one is a differential equation, the number of variables which can be initialized is 1 as discussed in Chapter 7. In order to solve this problem, the value of y was taken as the correct initial value while that of y' and z were regarded as initial estimates. These were used to determine the correct initial conditions.

2 Problem No 8 - PI Control of CSTR (Ref: Liang, 1985)

A CSTR as shown in Figure 9.1 is operated with feed composition, temperature and heat duty to the reactor known and with a first order irreversible, endothermic reaction.

$$A \xrightarrow{k} B$$

The PI controller is used to control the reactor temperature by adjusting the heat transfer rate through a steam coil.

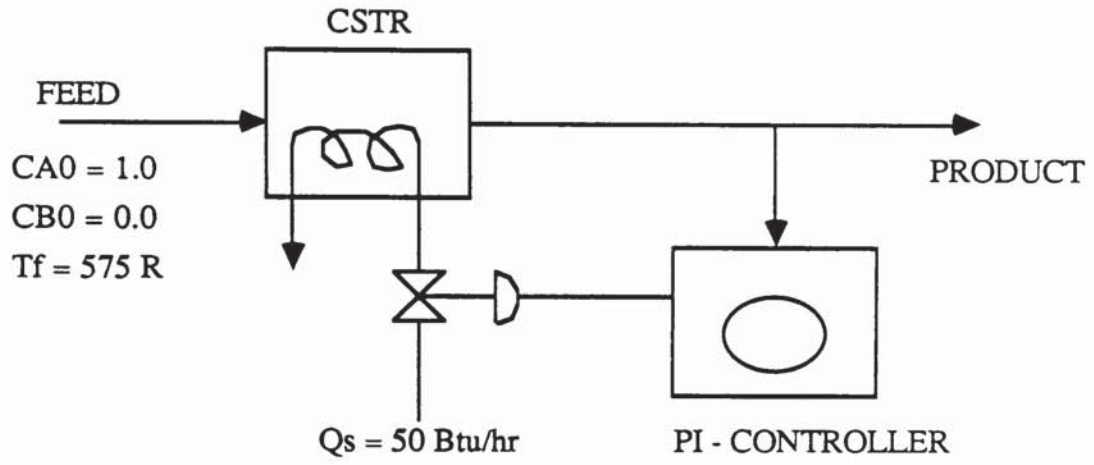


Figure 9.1 Process flowsheet of Problem 8

Initially the heat transfer rate from the steam coil is 50 Btu/hr and the temperature and composition in the reactor are the same as those of the feed stream. It is also assumed that no reaction was taking place initially, and the reaction is activated by a trace of catalyst at the start of simulation. The open-loop and closed loop response of the reactor will be simulated.

Model Equations:

The equations that describe the dynamics of the reactor are:

For temperature (T)

$$V.Cp.\frac{dT}{dt} = F.\,Cp.\,T_f - F.\,Cp\,.\,T - \Delta H\,.\,V.\,Rate + Q$$

For composition (C_A)

$$V\frac{dC_A}{dt} = F.C_{A0} - F.\,C_A - V.\,Rate$$

where

V is the volume of the reaction mixture
C_p is the heat capacity
F is the flowrate
ΔH is the heat of reaction
Q is the heat duty to the reactor
Rate is the reaction rate, expressed as Rate = kC_A

$$k \quad K \, EXP \left(\frac{-E}{RT} \right)$$

K is the frequency factor
E is the activation energy
R is the ideal gas constant

The equations for the controller are:

$$e = T_S - T$$

$$Q = Q_S \left(1 + K_C \left(e + \frac{1}{t_I} \int e \, dt\right)\right)$$

where

T_S is the reference temperature
Q_S is the reference or steady state heat duty
K_C is the controller gain
t_I is the integral time

To avoid computing the integral in the above equation, we introduce the variable X, defined as

$$X = \int e \, dt$$

or

$$\frac{dX}{dt} = e$$

Here X becomes the unit dependent variable and the controller equations become

$$e = T_S - T$$

$$\frac{dX}{dt} = e$$

$$Q = Q_S \left(1 + K_C \left(e + \frac{X}{t_I}\right)\right)$$

The complete set of variables in these equations are

C_A, T, Q, X

The parameters are

V, Cp, ΔH, K, E, R, Q_S, t_I, K_C, T_S

The forcing functions are

F, T_f, C_{A0}

Parameters:

V = 100 ft^3, Cp = 60 Btu/ft^3/°R	R = 1.987 Btu/lb-mole/°R
E = 35000 Btu/lb mole	K = 6.0 x 10^4 /hr
ΔH = 5000 Btu/lb-mole of A	T_S = 600°R
t_I = 0.16667 hr	Q_S = 50 Btu/hr

K_C = 0.0 (open-loop) and 0.5 (closed-loop)

T_f = 575°R	C_{A0} = 1.0 mole/ft^3
F = 60 ft^3/hr,	

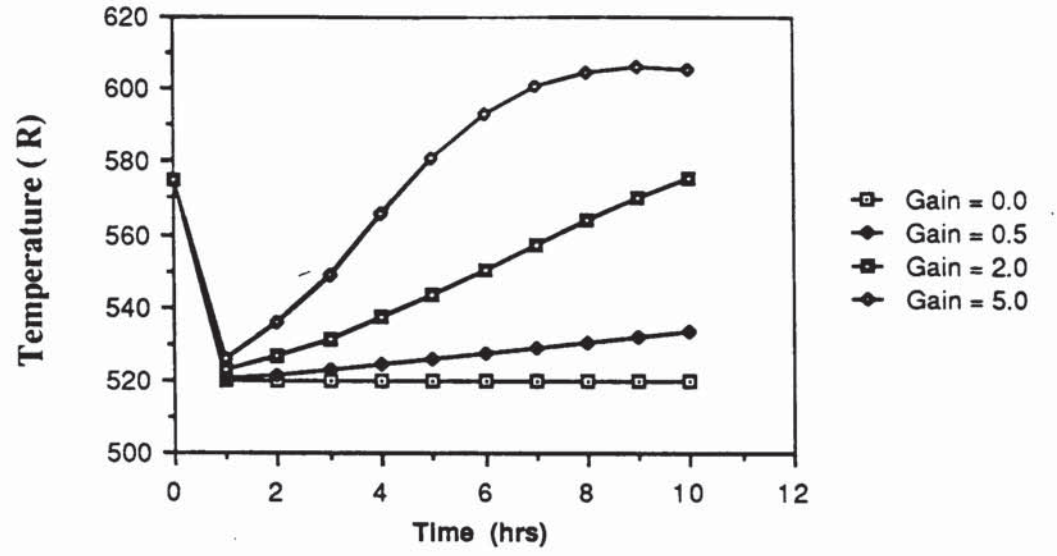
Initial conditions

C_A = 1.0 mole/ft^3	T = 575 °R
Q = 50 Btu/hr	X = 0.0

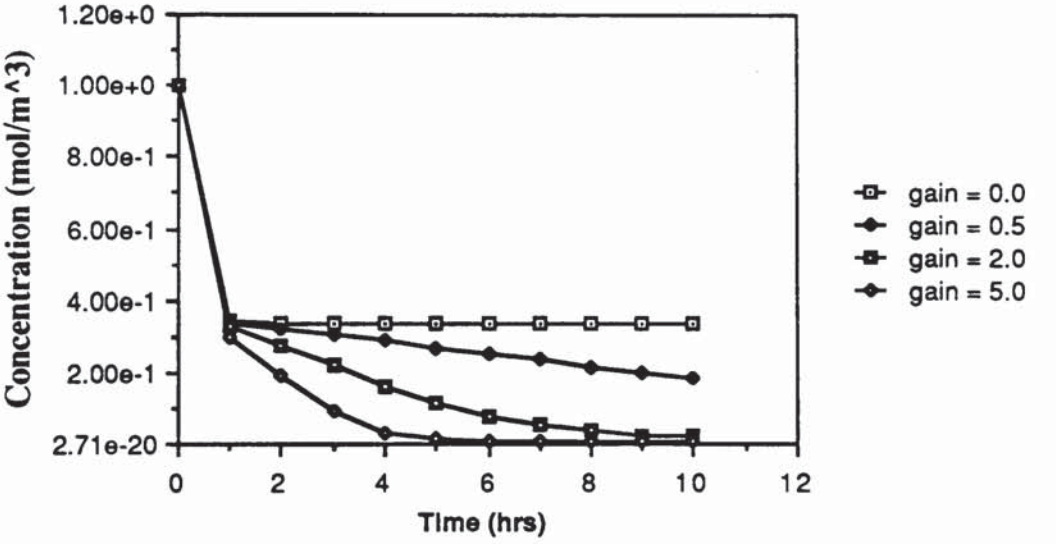
9.4.2 SOLUTION OF EXAMPLE TYPE III PROBLEMS

Problem numbers 7 and 8 were solved using the USRSUB option. The data files, and the USRSUB subroutines were prepared as described for Example type I problems. The first problem is a simple one, but it shows that a differential-algebraic system can be solved in much the same way as a differential system. The result of y at time 0.05 hrs is 0.5861, which compares very well with the exact solution of 0.5864 from the literature.

In Problem No 8, one open loop and 3 closed loop responses with gains of 0.5, 2.0 and 5.0 and a constant integral time of 0.16667 were simulated. The graph in Figure 9.2 shows the response of the temperature and the concentration to these conditions. With fixed integral time, of 0.16667 hrs and other parameters as shown in Section 9.4.1, four different runs were carried out using different gains.



a)



b)

Figure 9.2 - Plot of Temperature and Concentration versus time.

The response shown in Figure 9.2a for the reactor temperature and Figure 9.2b for the concentration of component A in the reactor is in agreement with that given by Liang (1985). It shows that the control is not as effective as it should be, especially at low gain values. The simulation did not start from steady state values of these parameters, which explains why high gain values are needed to bring it to steady state at the set point conditions. This in turn increases the temperature, which accelerates the reaction. Other forms of experimentations can be carried out.

9.5 EXAMPLE TYPE IV - DAE SYSTEMS USING DASP MODULES

This section demonstrates how the equations to be solved can be generated from the modules in DASP. Two examples are presented in the example problems described below.

9.5.1 DESCRIPTION OF METHANOL MIXER PROBLEM

1 Problem No 9 - Methanol Mixer Tank

(Ref: Thambynayagom et al, 1981; Smith and Morton, 1987)

The simulation concerns the response of a methanol mixer and two control loops to a series of imposed changes in feed flow rate. The event processing part of this problem is discussed in Example Type V in Section 9.6. The system consists of a perfectly stirred tank with two methanol solution feeds (concentrated and dilute), one water feed and one outlet stream. The tank acts as a perfect mixer with no heat transfer. The methanol solution streams are controlled by two-position actuator valves which can be fully on or fully off only. Control valves are placed on the water feed stream and the tank outlet stream. There are two control loops and the object is to keep the level in the tank at 0.8m and the mole fraction of methanol at 0.4 in the outlet stream, subject to a series of imposed system changes (see section 9.6). The flowsheet of this problem is shown in Figure 9.3, the corresponding block diagram in Figure 9.4 and a description of the units and streams in the block diagram is given in Table 9.2 (page 175).

The tank is modelled by the following equations:

$$\rho A \frac{dH}{dt} = \sum_{j=1}^{NIS} F_j - F_o$$

$$\rho A \left(H \frac{dX_k}{dt} + X_k \frac{dH}{dt}\right) = \sum_{j=1}^{NIS} F_j X^j_k - F_o X_k$$

where

NIS is the number of input streams

F_j is the flowrate of the j^{th} input stream

The controllers are modelled by PI-control equations. The water flow valve is modelled by a simple gain. The transmitter and the discharge valve are both modelled by first order dynamic equations.

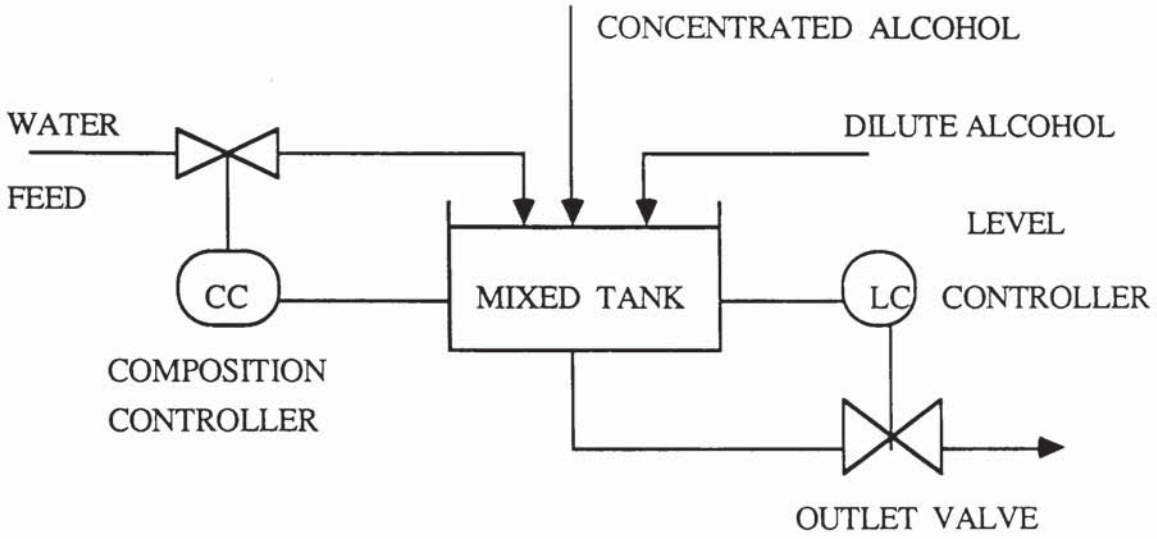


Figure 9.3 - Flowsheet of Methanol Mixer Problem

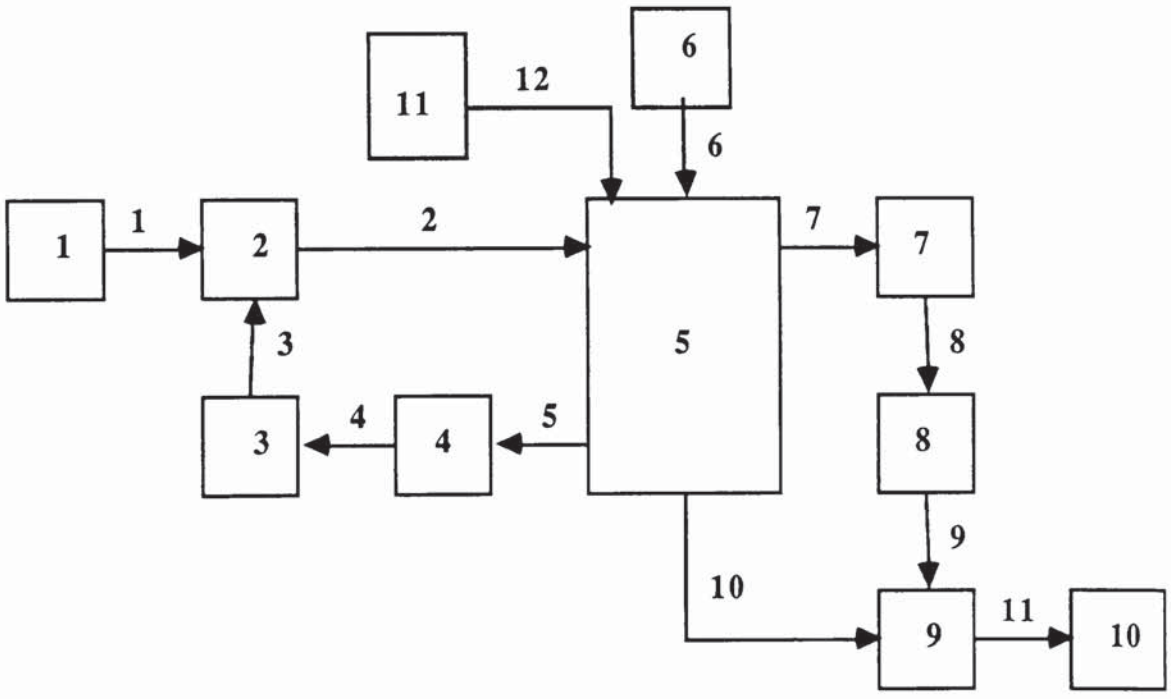


Figure 9.4 - Block Diagram of Methanol Mixer Problem

Initial values of variables and parameters

Water feeder

$X_W = 1.0,\ X_M = 0.0,\ P = 2.5E5$ Pa

Water feeder control valve

F_{min} = 0.0 kmol/hr, F_{max} = 1800 kmol/hr, F = 166.67kmol/hr

Gain = 1.0

Composition controller

K_p = 15.0, TI = 4.0 hrs, AXN = -1.0, X_{SET} = 0.4

Composition transmitter

K_p = 5.0, TAU = 0.003 hrs

Mixer Tank

T = 293K, P = 10^5 Pa, A = 1.7672 m^2

H = 0.8m, X_M = 0.5, X_W = 0.5, ρ_{mix} = 28.2933 kmol/m^3

Concentrated Methanol Feeder

X_M = 0.9, X_W = 0.1, F = 100 kmol/hr

Dilute methanol feeder

X_M = 0.0714, X_W = 0.9286, F = 70 kmol/hr

Level Controller

K_P = 400, TI = 5 hrs, AXN = -1.0, H_{SET} = 0.8m

Outlet Control Valve

K_P = 4, TAU = 0.0015 hr, F_{min} = 0.0 kmol/hr,

F_{max} = 2829.33 kmol/hr, FO = 0.0 kmol/hr

Outlet

FO = 0.0 kmol/hr, P = 10^5 Pa

9.5.2 SOLUTION OF METHANOL MIXER PROBLEM

The methanol-mixer problem is a good example to demonstrate the various features of DASP. Here it is shown how the flowsheet can be converted into the equivalent block diagram and the ability of assembling all the model equations from the DASP module library. The block diagram of this flowsheet is shown in Figure 9.4. DASP requires that

all streams originate from or terminate at a unit. Hence the water flow stream originates from FEED1, the dilute methanol flow stream from FEED2, the concentrated methanol stream from FEED3 and the outlet stream terminates at OUTLET. The water feeder control valve (VALVE1) was modelled by a simple gain, with a value of 1.0, the concentration controller (CONTROL1) by a PI control model while the measurement or transmitter (TRANSMIT) was modelled by a first order dynamic equation. The level controller (CONTROL2) was modelled by a PI-control model. The control valve was made up of two modules, a first order lag (MANIPUL) and a simple gain (VALVE2). This arrangement was necessary because the output of the 1st order lag model is a correction to flowrate, so the simple gain model generates the actual flow rate. This arrangement also assumes that the flowrate does not depend on the hydrostatic pressure in the tank. All the model equations were assembled from DASP modules, using the following routines:

SMGAIN for VALVE1,	CONTRL for CONTROL1
ORDERN for TRANSMIT,	MIXTK1 for Mixed Tank
CONTRL for CONTROL2,	ORDERN for MANIPUL
SMGAIN for VALVE2	

All the modules are in the library except for SMGAIN and MIXTK1 which were specially written for this problem. The data files, CINDAT9, CTOPOL9, CUNIT9 and COMDT9 were prepared as the general input data, topology data, initial values and components data respectively. Using the topology data file, a Fortran subroutine was generated, which contains calls to the routines enumerated above from the DASP library. This program was compiled and linked with the other routines in DASP to produce an executable file, DASP.EXE on the IBM PC AT. The result of the simulation for 3 hrs is given in Appendix A. The liquid height in the mixed tank and the concentration of methanol were plotted against time. This is shown in Figure 9.5. It can be seen that the level controller is more effective in driving the level to the setpoint than the controller in driving the composition of methanol to its setpoint.

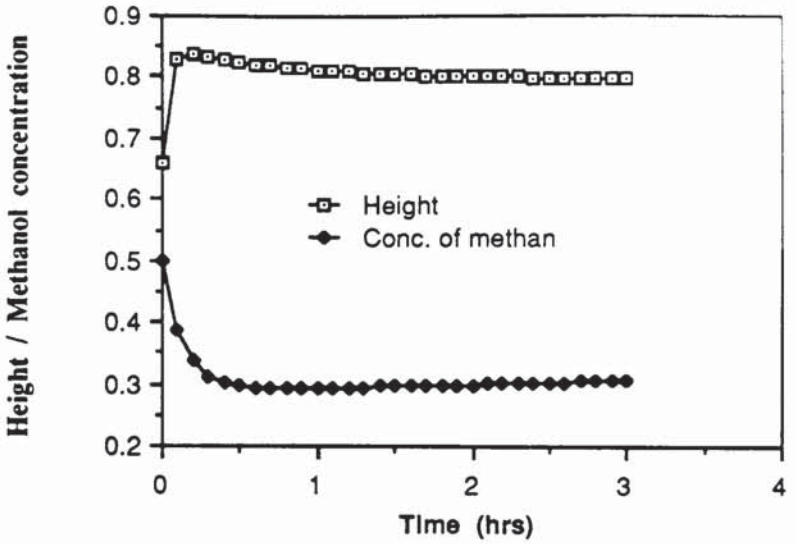


Figure 9.5 - Plot of height and Methanol concentration versus time

9.5.3 DESCRIPTION OF MIXER SPLITTER PROBLEM

1 Problem No 10 - Mixer-Splitter System

(Ref: Husain, 1986, p211)

The system consists of a mixer and a splitter as shown in Figure 9.6. The feed stream consists of three components, namely butane, propane and ethane at 20atm pressure and 30°F. It enters a mixer with a holdup of 200 lbmoles. The mixer output is then split in a splitter, 80% of it going out as stream 3 and 20% recycling (Stream 4). The initial composition of all the streams are equal with equimolal fractions of the three components.

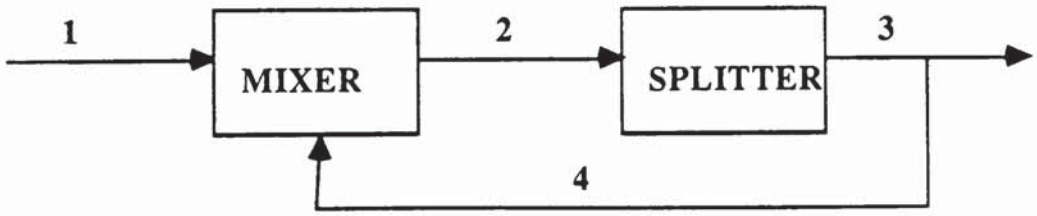


Figure 9.6 - Mixer-splitter system flowsheet

The flowrate of stream 1 is 100 lbmol/hr. At time 0^+, a step increase in the molefraction of ethane occurs, raising it from 0.3334 to 0.4, and at the same time the concentration of

butane decreases to 0.26 in the input. What happens to the butane concentration in stream 2?

This problem can be solved by DASP by first preparing the block diagram of the flowsheet and the input data files. The block diagram is shown in Figure 9.7.

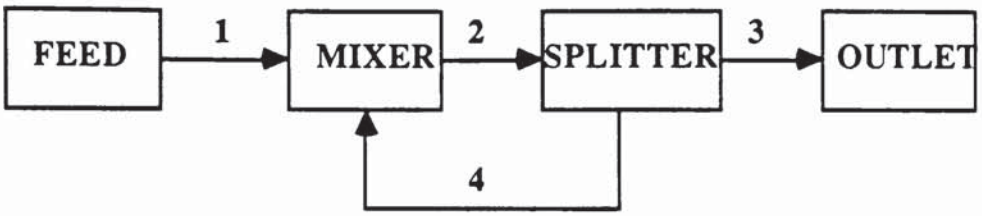


Figure 9.7 - Block diagram of mixer-splitter system

The following data were used to prepare the data files.

Steady State Conditions

Concentrations: (in all streams)

X_b = 0.3333 (mole fraction of butane)

X_p = 0.3333 (mole fraction of propane)

X_e = 0.3334 (mole fraction of ethane)

Feed

T = 30°F F = 100 lbmol/hr

Mixer

F = 124.9984 lbmol/hr T = 29.9998°F

M = 200 lbmole

Splitter

F_3 = 99.9935 lbmol/hr F_4 = 24.9984 lbmol/hr

T = 29.9998°F $\alpha_3 = 0.8$, $\alpha_4 = 0.2$

Modelling Equations

The mixer was modelled with the assumption of ideal mixing, volume of material in tank being constant, instantaneous pressure transmission, single phase and molar density and specific heat for all components being equal.

Overall mass balance

$$F_2 = F_1 + F_4$$

Mass balance for component i, for i = 1 to 3

$$M \frac{dX_{i2}}{dt} = \sum_{j=1}^{2} X_{ij} F_j - X_{i2} F_2$$

where j=1 refers to stream 1 and j=2 refers to stream 4.

Energy balance considering only sensible heat effect

$$M \frac{dT_2}{dt} = \sum_{j=1}^{2} F_j T_j - F_2 T_2$$

The splitter is modelled with no hold-up and no time lag. All output streams are assumed to have the same properties as the input stream, only the flowrates being different. Thus the model consists of only an overall balance for each output stream given a constant split fraction.

$$F_3 = F_2 * \alpha_3$$

$$F_4 = F_2 * \alpha_4$$

At time 0^+ a step change was made in unit 1 (ie. FEED) in the values of the concentration as follows:

$$X_b = 0.26 \quad , \quad X_p = 0.34 \quad , \quad X_e = 0.4$$

This simple problem illustrates the usefulness of the equation-oriented approach, in handling recycling.

9.5.4 SOLUTION OF MIXER SPLITTER PROBLEM

The mixer-splitter problem was modelled by a mixed tank and a splitter using routines from DASP library. This example shows a straight-forward translation of the flowsheet

to the equivalent block diagram, with the addition of an input and an output unit. The problem was simulated for 2 hrs and the result is given in Appendix A. A plot of the composition of butane against time is shown in Figure 9.8.

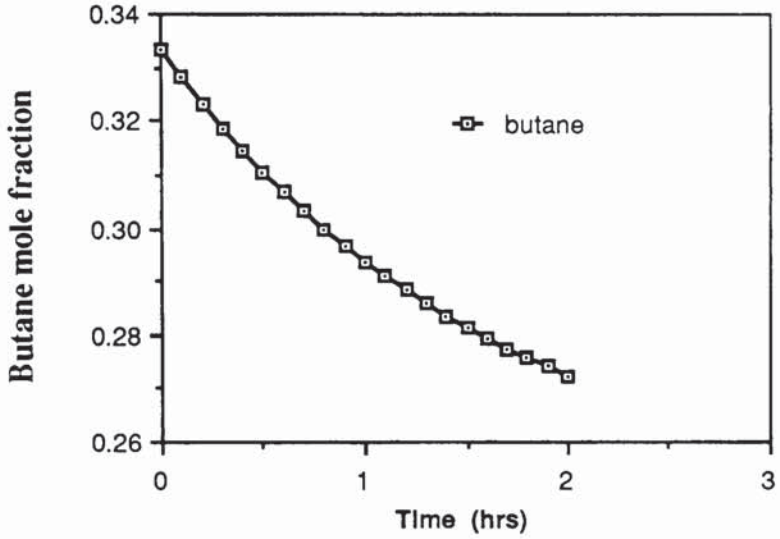


Figure 9.8 - Plot of Butane Concentration versus Time.

9.6 EXAMPLE TYPE V - SPECIAL FEATURES OF DASP

This section shows how DASP can be used to handle events, use the upset functions to vary the feed variables, modify a flowsheet and insert breakpoints during the interval of integration. In all cases, problems No 8 and 9 will be used for the USRSUB and DASP module options respectively.

9.6.1 DEMONSTRATION USING USRSUB OPTION

(a) Event Processing Feature

An event code which consisted of changing the value of the inlet concentration of component A, CA0, was coded in the USRSUB routine event section (JS=4). A time event was set up to occur at time 0.2 hrs. Control was returned to the user at this time and the event processing package called. The event section of the USRSUB routine was called to execute the event code. The result is given in Appendix A.

(b) RERUN Feature

At the end of the final time of simulation, the problem was restarted after modifying the inlet composition from 1.0 mole/ft^3 to 2.0 mole/ft^3.

(c) Breakpoint Feature

Breakpoints were set along the interval of integration so that control returns to the user after every 0.1 hrs. The values of the output variables were viewed and at time equal to 0.5 hrs, the inlet concentration of Component A was modified from 1.0 mole/ft^3 to 2.0 mole/ft^3.

(d) The use of the Upset Functions

Using the event feature, at time equal to 0.2 hrs control was returned to the user and the pulse function was used to upset the value of the inlet concentration of component A, so that at time 0.2 hrs, it changes from 1.0 to 3.0 mole/ft^3 and stays there until 0.4 hrs and then dropped back to its original value of 1.0 mole/ft^3. The results of these investigations are shown in the plot of Figure 9.9. It is possible to combine these features to create new special features.

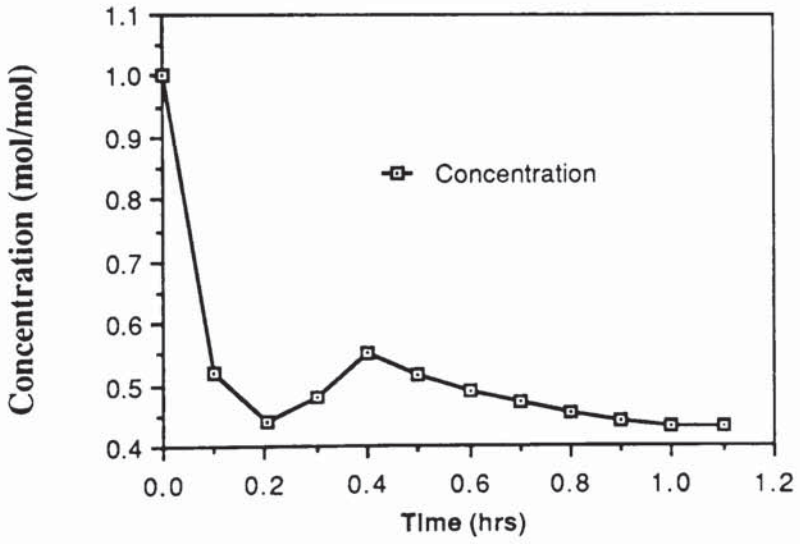


Figure 9.9 - Plot of Concentration versus Time (Event Option)

9.6.2 DEMONSTRATION USING MODELS FROM DASP

(a) Event Processing Feature

In this problem, the methanol solution streams are controlled by an actuator valve which can be switched fully on or off only. Since there was no actuator valve available in the Library, the event processing feature of disconnecting a unit or reconnecting a unit from the flowsheet was used. The sequence of operation of the flowsheet was as follows:

(i) Run with methanol solution feeds off until the mole fraction of methanol reaches 0.4.

(ii) Run with concentrated stream on, dilute stream off for 5 minutes.

(iii) Run with both solution feeds on for 10 minutes.

To start the simulation, all the units were initially connected to the flowsheet. At the time zero, the two methanol feed units were disconnected from the flowsheet and the simulation proceeded until the concentration of methanol reached 0.4. The event processing routines determined the exact time when the concentration reached 0.4. Then control was passed to the event processing package, where the concentrated methanol stream was reconnected and the new process simulated for 5 minutes. Then using the event processing package again, the dilute methanol unit was reconnected and the process simulated for 10 minutes. A plot of the concentration against time is shown in Figure 9.10. This example clearly demonstrates the powerful feature of DASP with regard to event processing. This operation can also be carried out using a combination of the break point facility and the MODIFY routine. This can be done as follows: At the initial time, using the MODIFY routine, the flow rates of the methanol feeds will be set to zero. After the exact time when the methanol concentration has reached 0.4 has been determined, the flow rate of the concentrated feed is reset to its former value. When finally control returns to the user after 5 minutes from the time of last event time, the flow rate of the dilute feed is reset to its original value and simulation proceeds for another 10 minutes. This alternative procedure was also carried out successfully, with the same results.

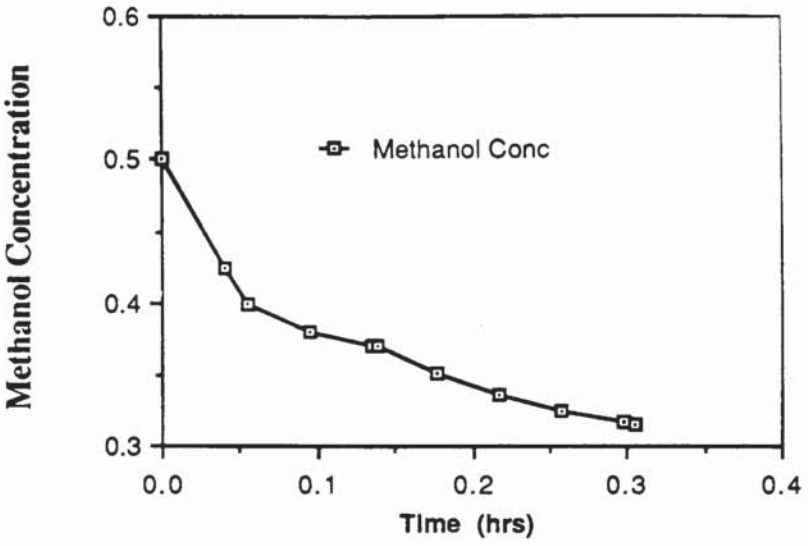


Figure 9.10 - Plot of Methanol Concentration versus time (event)

(b) The use of the Upset Functions

The aim here is to investigate what happens to the methanol concentration in the mixed tank if there is a change in the flowrate of the dilute methanol solution from 70 kmol/hr to 100 kmol/hr at the end of the last event operation. The result of this investigation is shown in Figure 9.11.

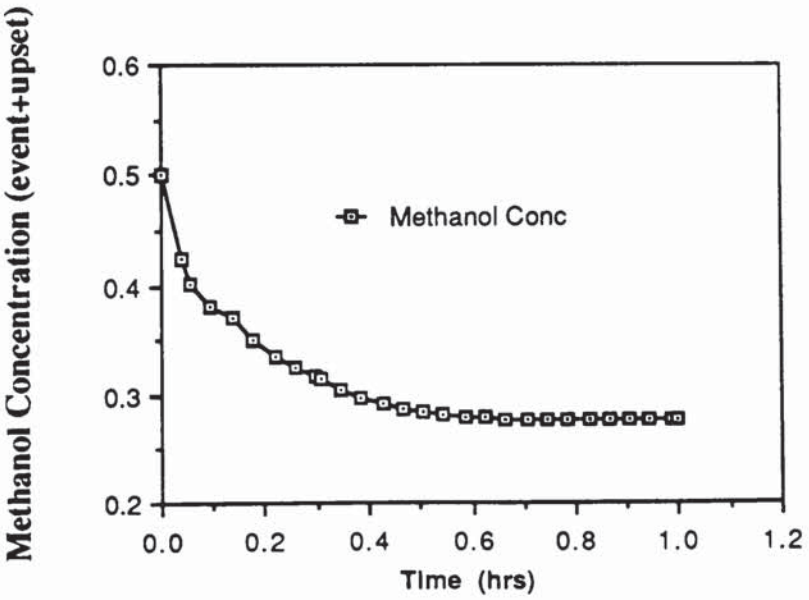


Figure 9.11 - Plot of Methanol Concentration versus time (event + upset)

Table 9.2 - Description of Block Diagram in Figure 9.4

UNITS	STREAMS
1. Inlet for water	1. Water feed
2. Inlet control valve	2. Water feed after inlet valve
3. Composition controller	3. Control signal to inlet valve
4. Concentration measurement	4. Measured signal to controller
5. Mixer tank	5. Measurement signal
6. Dilute alcohol tank	6. Dilute alcohol stream
7. Level Controller	7. Signal to controller
8. Valve actuator	8. Controller signal to valve actuator
9. Outlet valve	9. Actuator signal to outlet valve
10. Outlet unit	10. Mixer tank outlet
11. Concentrated alcohol tank	11. Outlet valve stream
	12. Concentrated alcohol stream.

9.7 CONCLUSION

In this chapter, 10 different examples have been used to demonstrate the capabilities of DASP, including solution of algebraic, differential and differential-algebraic equation systems. The features demonstrated include event handling capabilities, use of the upset functions, the MODIFY routine and the setting of break points along the interval of integration. These examples clearly show that DASP is a very powerful tool in dynamic simulation. The main limitation on results of DASP is the availability of suitable data in the literature for comparison with other simulation systems or plant data.

CHAPTER 10

DISCUSSION AND CONCLUSIONS

10.1 INTRODUCTION

The modelling of the transient behaviour of chemical plants results in differential-algebraic equations (DAEs). Dynamic simulation is concerned with solving these equations using numerical methods. The data generated during this process can be useful in assessing the performance of the plant with regard to its stability during start-up and shut down, investigating how fluctuations in the values of the feed stream variables affect variables of interest in the plant and so on. However, starting from the modelling stage, through the input data preparation, the solution of the DAEs and on to the final analysis of the results special techniques and methods are required depending on the problem. These techniques are limited in number, although they cover a large range of problems. This makes it possible to construct a general purpose process dynamic simulator. The advantage of such a simulator is that it relieves the user of the burdens of programming the special techniques required, giving time to concentrate on the engineering part of the whole activity, the choice of models to use and data to describe the problem.

Of the various approaches to the design of dynamic simulators, the equation-oriented approach is best as it not only solves all the problems encountered by other methods with regard to design, optimization and flow-pressure modelling, but also offers complete flexibility with regard to the formulation of the equations and solution of these equations. There are a series of problems which must be solved in order to implement this approach. The implementation of an equation-oriented approach for dynamic simulation studies requires the following:

1. The use of techniques to reduce the large computer memory required for storing the large system of equations and solution data generated.

2. Complex algorithms for solving the differential-algebraic equations of the user's problem.

3. A method for providing consistent and correct initial values for all the variables and their derivatives in the model equations.

4. A suitable executive structure, which is effective to carry out the tasks of directing, controlling and coordinating the functions of the other parts of the program package.

5. Development of a methodology for structuring the equations to be solved in modules in such a way that these equations can easily be assembled from the modules to form the equations of the user's problem as required.

6. A suitable formulation strategy of the equations in the model routines so that the same equations can be used for both steady state and dynamic simulations and optimisation.

7. A technique for detecting a state event and finding the exact time of occurence of the event.

This work has developed a mechanism for gathering together recent algorithms and methods from various disciplines to design DASP as a dynamic simulator which fulfills the requirements mentioned above.

10.2 THE STRUCTURE OF DASP

The CSSL-type structure used in the Executive progam is simple and flexible and offers

the advantage that several new regions and subsections could be created for other purposes, such as an optimization section within the Dynamic region. This can be done by defining new values of JSECTN, the section control variable. The use of an interface routine as a bridge between the Executive and other parts of the package makes it possible to remove or add new equation solvers or event processing routines to the package. The structure of DASP has redefined and extended the CSSL structure by providing new sections within the Dynamic region for steady state and dynamic simulations, perturbation and event processing sections. The sectional structure of DASP's Executive makes it easy to debug and maintain.

10.3 THE SOLUTION OF DAES

The solution of DAEs by ODE methods has made equation-oriented simulation a viable method and a better alternative to the sequential modular approach. It is now possible to solve implicitly defined DAEs using such codes as DASSL (Petzold, 1983). However, most of the available dynamic simulators only solve a restricted form of DAEs. DASP imposes no such restriction but uses a general model formulation technique discussed in Chapter 2. There are still problems to be solved with regard to the use of ODE methods in solving DAEs as discussed in Chapter 3. It is hoped that new techniques will be developed to solve these problems, which occur when using dynamic simulation for design.

10.4 THE INITIALIZATION OF DAES

One of the problems with the solution of DAES using ODE methods such as the BDF is that the algorithm requires the correct initial conditions for both the differential and algebraic variables and their derivatives. The original DASSL program used a method whereby, given some of the initial values of the derivatives and all the values of the variables, the rest of the values of the derivatives were estimated using the Euler method coupled to a damped Newton iteration. This did not always work. In DASP a method of initialization is used which provides for the correct initial values for all the variables

and derivatives. The present implementation assumes that the user provides the correct initial values of the differential variables as in ODE systems and an estimate of the algebraic variables. The correct initial values are then computed by solving a nonlinear system in which the derivatives of the differential variables and the algebraic variables form the vector of unknown variables. However, in theory any subset of the differential and algebraic variables and the derivatives of the differential variables could be the N unknown variables for the N equations in the model to be solved.

10.5 ORGANIZATION OF EQUATIONS IN MODULES

The use of in-built model routines in an equation-oriented environment eliminates the need for the user to write model routines. But this presents problems to the simulator with regard to assembling the equations from the relevant modules for the equation solvers. DASP solves this using a combination of the incidence and connection matrices which are defined in the context of the module structure. While in simulators such as DPS (Wood et al, 1984), several types of variables and elements are defined, which the user must know in order to use the simulator, DASP introduced the unit module concept, which is defined as part or whole of a unit operation, in which only one occurence of a recognised variable type is permitted. This makes it possible for all the modules to have a common structure. The simulator does not need to know whether the module is a controller or a distillation column. It also makes it easier to write new modules. With the structuring of the modules into sections, new sections can be added, by introducing new values for the section control variable, JS. The main limitation of the present design is that the total number of variables in the system is arbitrarily divided into parameters and variables and uses integer control variables to offer the user different choices of variables and parameter sets. This is unnecessary, as explained in Appendix C, where a development is described which will eliminate this limitation.

10.6 EVENT PROCESSING

State and time events are handled by DASP using simple methods (discussed in Chapters

3 and 8) which accurately detect state events and determine the event times. Although only events whose sequence of occurrence are known in advance are accommodated at present, it is a simple matter extending it to multiple events which can occur in any sequence as described in Chapter 8 and Appendix C. Also the MODIFY routine offers a number of features which make dynamic simulation an effective tool. Thus the value of any parameter or variable can be viewed or changed during the dynamic region and the upset functions can be used to perturb the variables or parameters of any module.

10.7 SPECIAL FEATURES OF DASP

DASP offers the user the ability to re-run a simulation starting from any point during the simulation, modify a given flowsheet by disconnecting or reconnecting streams and units. This task makes use of the incidence and connection matrices of the problem and checks for consistency before restarting. It also provided a way of handling latency where some units in the flowsheet are dynamically inactive as is the case with startup and shutdown of chemical processes and in batch operations. The fact that it can run on a variety of computers, especially microcomputers which are readily available, with only trivial modifications, such as changing the default input and output unit numbers makes DASP very portable.

10.8 DEMONSTRATION OF DASP

The example problems solved in Chapter 9 demonstrate the capabilities of DASP in terms of solving DAEs, handling recycles, state and time event processing and latency. The MODIFY routine was used to set breakpoints along the interval of integration to demonstrate pausing to view or plot the values of the variables before continuing the simulation. It was also shown how the upset functions can be used to perturb the system by introducing disturbances in the values of the feed stream variables. All these show that the features of DASP described in this thesis work. The results of the problems solved agree with literature results where available.

10.9 LIMITATIONS AND FUTURE WORK

There are, however, a number of areas which need further investigation. Firstly, a database management system is needed, not only for data transfer and storage, but to be used in conjunction with a suitable graphics package to develop an input language. This would act as the front-end of the package and will be useful for generating a problem flowsheet interactively, to input and output data and information to and from the program via the database. Also graphical forms of output can be generated as well. At present, numbers are used to communicate between the user and the simulator and this is inconvenient. But by using a pointing device to point to options on the screen, the user interface could be improved. The modification of the flowsheet can be done interactively if the above-mentioned language and graphics are added. It will be useful to use multiple windows to display results and error messages. DASP already provides a means of doing this by providing different logical unit numbers for the different channels of communication such as input, output, error message and graphics.

Another very important area is the development of more modules such as distillation columns, absorption column and chemical reactors. More realistic chemical plant modules are needed to test all the capabilties of DASP and to effect any changes necessary.

Also the problem of finding the initial quesses and a global solution method for converging the nonlinear equations for steady state simulation are left as future work.

In summary, all these developments can be added to a program, which has been shown to be robust and portable and whose structure and design features were constructed in such a way that these new developments can easily be added.

APPENDIX A

RESULTS OF EXAMPLE PROBLEMS

PROBLEM NO 1

TITLE: PROBLEM NO 1, MICHELSEN 1976

STATISTICS OF THE SIMULATION

NO OF FUNCTION EVALUATIONS : 563
NO OF JACOBIAN EVALUATIONS : 39
NO OF STEPS TAKEN : 157
NO OF ERROR TEST FAILURES : 12

RESULTS OF SIMULATION

S/N	TIME HR	VARIABLE 1	VARIABLE 2	VARIABLE 3	VARIABLE 4
1	.00000D+00	0.75917D+03	0.00000D+00	0.60000D+03	0.10000D+00
2	.10000D+04	0.12107D+04	0.12746D-03	0.12083D+04	0.43859D-03

PROBLEM NO 2

TITLE: PROBLEM NO 2, REF. CHAN ET AL 1978

STATISTICS OF THE SIMULATION

NO OF FUNCTION EVALUATIONS : 344
NO OF JACOBIAN EVALUATIONS : 25
NO OF STEPS TAKEN : 123
NO OF ERROR TEST FAILURES : 10

RESULTS OF SIMULATION

S/N	TIME HR	VARIABLE 1	VARIABLE 2
1	.00000D+00	0.10000D+01	0.00000D+00
2	.50000D+02	0.42847D-03	0.13898D-01

PROBLEM NO 3

TITLE: PROBLEM NO 3, REF. BUI 1981

STATISTICS OF THE SIMULATION

NO OF FUNCTION EVALUATIONS : 889
NO OF JACOBIAN EVALUATIONS : 56
NO OF STEPS TAKEN SO FAR : 273
NO OF ERROR TEST FAILURES : 30

RESULTS OF SIMULATION

S/N	TIME HR	VARIABLE 1 ----	VARIABLE 2 ----	VARIABLE 3 ----
1	.00000D+00	0.30000D+01	0.10000D+01	0.20000D+01
2	.30000D+03	0.48312D+01	0.12584D+01	0.31688D+01

PROBLEM NO 4

TITLE: PROBLEM NO 4, MICHELSEN 1976 PROBL 1

STATISTICS OF THE SIMULATION

NO OF FUNCTION EVALUATIONS : 394
NO OF JACOBIAN EVALUATIONS : 51
NO OF STEPS TAKEN : 92
NO OF ERROR TEST FAILURES : 4

RESULTS OF SIMULATION

S/N	TIME HR	VARIABLE C1 ----	VARIABLE C2 ----	VARIABLE C3 ----
1	.00000D+01	0.10000D+01	0.00000D+00	0.00000D+00
2	.10000D+01	0.96646D+00	0.30747D-04	0.33507D-01
3	.20000D+01	0.94163D+00	0.27020D-04	0.58345D-01
4	.30000D+01	0.92191D+00	0.24386D-04	0.78067D-01
5	.40000D+01	0.90554D+00	0.22407D-04	0.94435D-01
6	.50000D+01	0.89154D+00	0.20855D-04	0.10844D+00
7	.60000D+01	0.87930D+00	0.19598D-04	0.12068D+00
8	.70000D+01	0.86840D+00	0.18552D-04	0.13158D+00
9	.80000D+01	0.85857D+00	0.17666D-04	0.14141D+00
10	.90000D+01	0.84962D+00	0.16903D-04	0.15036D+00
11	.10000D+02	0.84139D+00	0.16236D-04	0.15859D+00

PROBLEM NO 8 (EVENT OPTION)

VALUES OF PARAMETERS IN THE SYSTEM

INLET FLOWRATE = 0.60000D+02 ft^3/hr
INLET TEMPERATURE = 0.57500D+03 R
INLET CONCENTRATION = 0.10000D+01 lbmole/ft^3

CONTROLLER SETPOINT = 0.60000D+03 R
CONTROLLER GAIN = 0.10000D+01 (Btu/hr)/R
INTEGRAL TIME (HR) = 0.16667D+00 hr

STATISTICS OF THE SIMULATION

NO OF FUNCTION EVALUATIONS : 576
NO OF JACOBIAN EVALUATIONS : 70
NO OF STEPS TAKEN : 106
NO OF ERROR TEST FAILURES : 18

RESULTS OF SIMULATION

Total number of variables: 4
Total number of equations: 4

S/N	TIME HR	TEMPERATURE R	CONCENTRATN MOL/M^3	HEAT LOAD BTU/HR
1	.00000D+00	0.57500D+03	0.10000D+01	0.50000D+02
2	.10000D+00	0.53518D+03	0.52146D+00	0.49149D+04
3	.20000D+00	0.52833D+03	0.43811D+00	0.73200D+04
4	.30000D+00	0.52482D+03	0.47981D+00	0.97075D+04
5	.40000D+00	0.52192D+03	0.55357D+00	0.12162D+05
6	.50000D+00	0.51994D+03	0.51726D+00	0.14637D+05
7	.60000D+00	0.51883D+03	0.49053D+00	0.17111D+05
8	.70000D+00	0.51827D+03	0.47061D+00	0.19583D+05
9	.80000D+00	0.51805D+03	0.45508D+00	0.22050D+05
10	.90000D+00	0.51804D+03	0.44235D+00	0.24510D+05
11	.10000D+01	0.51815D+03	0.43134D+00	0.26963D+05
12	.10000D+01	0.51815D+03	0.43134D+00	0.26963D+05

PROBLEM NO 9.1 (SIMULATION OPTION)

TITLE: PROBLEM NO 9, METHANOL-MIXER SYSTEM

TOPOLOGY INFORMATION

TOTAL NO OF UNITS	TOTAL NO OF STREAMS
11	12

UNITS DATA

S/N ***	UNIT NAME	UNIT NO	UNIT CODE	UNIT TYPE
1	FEED1	1	0	0

2	VALVE1	2	22	1
3	CONTROL1	3	1	1
4	TRANSMIT1	4	4	1
5	MIXED-TANK	5	11	1
6	FEED2	6	0	0
7	CONTROL2	7	1	1
8	TRANSMIT2	8	4	1
9	VALVE2	9	22	1
10	OUTLET	10	0	0
11	FEED3	11	0	0

STREAMS DATA
==========

S/N ***	STRM NAME ---	STRM NO ---	UNIT FROM ---	UNIT TO ---	STRM TYPE ---
1	S1	1	1	2	3
2	S2	2	2	5	3
3	S3	3	3	2	0
4	S4	4	4	3	0
5	S5	5	5	4	0
6	S6	6	6	5	3
7	S7	7	5	7	0
8	S8	8	7	8	0
9	S9	9	8	9	0
10	S10	10	5	9	3
11	S11	11	9	10	3

UNITS CONNECTION MATRIX
****

UNIT NO -------	UNITS CONNECTED TO IT ----------		
1	2		
2	1	5	3
3	4	2	

4	5	3				
5	2	6	11	9	4	7
6	5					
7	5	8				
8	7	9				
9	5	10	8			
10	9					
11	5					

STATISTICS OF THE SIMULATION

==========

NO OF FUNCTION EVALUATIONS : 902
NO OF JACOBIAN EVALUATIONS : 51
NO OF STEPS TAKEN : 143
NO OF ERROR TEST FAILURES : 10

RESULTS OF SIMULATION

****

Total number of variables: 11
Total number of equations: 11

---

* UNIT NO = 2 MODULE NAME = VALVE1 *

****

S/N	TIME HR	LRATE kgmol/hr
1	.00000D+00	0.16667D+03
2	.10000D+00	0.16611D+03
3	.20000D+00	0.16204D+03
4	.30000D+00	0.15963D+03
5	.40000D+00	0.15803D+03
6	.50000D+00	0.15685D+03
7	.60000D+00	0.15588D+03
8	.70000D+00	0.15501D+03
9	.80000D+00	0.15420D+03
10	.90000D+00	0.15342D+03
11	.10000D+01	0.15265D+03
12	.11000D+01	0.15190D+03
13	.12000D+01	0.15116D+03
14	.13000D+01	0.15042D+03
15	.14000D+01	0.14969D+03
16	.15000D+01	0.14896D+03
17	.16000D+01	0.14824D+03
18	.17000D+01	0.14753D+03
19	.18000D+01	0.14681D+03
20	.19000D+01	0.14611D+03
21	.20000D+01	0.14541D+03
22	.21000D+01	0.14471D+03
23	.22000D+01	0.14402D+03
24	.23000D+01	0.14333D+03
25	.24000D+01	0.14265D+03
26	.25000D+01	0.14197D+03
27	.26000D+01	0.14130D+03

28	.27000D+01	0.14063D+03
29	.28000D+01	0.13997D+03
30	.29000D+01	0.13931D+03
31	.30000D+01	0.13866D+03

* UNIT NO = 5 MODULE NAME = MIXED-TANK *

***

11	.10000D+01	0.32371D+03
12	.11000D+01	0.32285D+03
13	.12000D+01	0.32201D+03
14	.13000D+01	0.32119D+03
15	.14000D+01	0.32038D+03
16	.15000D+01	0.31958D+03
17	.16000D+01	0.31880D+03
18	.17000D+01	0.31802D+03
19	.18000D+01	0.31726D+03
20	.19000D+01	0.31651D+03
21	.20000D+01	0.31577D+03
22	.21000D+01	0.31503D+03
23	.22000D+01	0.31431D+03
24	.23000D+01	0.31359D+03
25	.24000D+01	0.31288D+03
26	.25000D+01	0.31218D+03
27	.26000D+01	0.31149D+03
28	.27000D+01	0.31080D+03
29	.28000D+01	0.31012D+03
30	.29000D+01	0.30944D+03
31	.30000D+01	0.30878D+03

PROBLEM NO 9.2 (EVENT OPTION)

TITLE: PROBLEM NO 9, METHANOL-MIXER SYSTEM

EVENTS INFORMATION

****

Total number of events (NEVENT): 4

User-supplied events names (CENAME(*)):

TIME EVENT 1	STATE EVENT	TIME EVENT 2	TIME EVENT 3
Code numbers of events (KECODE(*)):			
1	2	1	1
Event times (EVTIME(*)) (HR):			
0.00000D+00	0.00000D+00	0.83000D-01	0.16670D+00
Units nos of variables (KEUNIT(*)):			
1	5	1	1
Stage or section numbers (KSUNIT(*)):			
0	0	0	0
Code number of variables (KVARNO(*)):			
0	21	0	0
Variables threshold values (TRSVAL(*)):			
0.00000D+00	0.40000D+00	0.00000D+00	0.00000D+00
Threshold tolerances (TRSTOL(*)):			
0.10000D-04	0.10000D-04	0.10000D-04	0.10000D-04
Threshold crossing directns (IDCROS(*)):			
1	-1	1	1

STATISTICS OF THE SIMULATION

==========

NO OF FUNCTION EVALUATIONS : 1232
NO OF JACOBIAN EVALUATIONS : 70
NO OF STEPS TAKEN : 184
NO OF ERROR TEST FAILURES : 21

RESULTS OF SIMULATION
****

Total number of variables: 11
Total number of equations: 11

---

* UNIT NO = 2 MODULE NAME = VALVE1 *
****

S/N	TIME HR	LRATE kgmol/hr
1	.00000D+00	0.16667D+03
2	.40000D-01	0.16904D+03
3	.54977D-01	0.16724D+03
4	.94977D-01	0.16538D+03
5	.13498D+00	0.16469D+03
6	.13798D+00	0.16464D+03
7	.17798D+00	0.16311D+03
8	.21798D+00	0.16178D+03
9	.25798D+00	0.16070D+03
10	.29798D+00	0.15981D+03
11	.30468D+00	0.15968D+03

* UNIT NO = 5 MODULE NAME = MIXED-TANK *
****

S/N	TIME HR	HEIGHT m	XCOMP kgmol/kgmol	XCOMP kgmol/kgmol
1	.00000D+00	0.66100D+00	0.50000D+00	0.50000D+00
2	.40000D-01	0.71131D+00	0.42394D+00	0.57606D+00
3	.54977D-01	0.72067D+00	0.40000D+00	0.60000D+00
4	.94977D-01	0.77006D+00	0.37966D+00	0.62034D+00
5	.13498D+00	0.79200D+00	0.37163D+00	0.62837D+00
6	.13798D+00	0.79287D+00	0.37111D+00	0.62889D+00
7	.17798D+00	0.82729D+00	0.35134D+00	0.64866D+00
8	.21798D+00	0.83848D+00	0.33615D+00	0.66385D+00
9	.25798D+00	0.84068D+00	0.32483D+00	0.67517D+00
10	.29798D+00	0.83992D+00	0.31632D+00	0.68368D+00
11	.30468D+00	0.83968D+00	0.31512D+00	0.68488D+00

* UNIT NO = 9 MODULE NAME = VALVE2 *
****

S/N	TIME HR	LRATE kgmol/hr
1	.00000D+00	0.27371D+03
2	.40000D-01	0.12300D+03
3	.54977D-01	0.13659D+03
4	.94977D-01	0.21099D+03
5	.13498D+00	0.24645D+03
6	.13798D+00	0.24786D+03
7	.17798D+00	0.30307D+03
8	.21798D+00	0.32405D+03
9	.25798D+00	0.33043D+03

10	.29798D+00	0.33190D+03
11	.30468D+00	0.33195D+03

PROBLEM NO 9.3 (EVENT + STEP FUNCTION)

TITLE: PROBLEM NO 9, METHANOL-MIXER SYSTEM

STATISTICS OF THE SIMULATION

==========

NO OF FUNCTION EVALUATIONS : 1518
NO OF JACOBIAN EVALUATIONS : 84
NO OF STEPS TAKEN : 238
NO OF ERROR TEST FAILURES : 26

RESULTS OF SIMULATION

****

Total number of variables: 11
Total number of equations: 11

---

* UNIT NO = 2 MODULE NAME = VALVE1 *

****

S/N	TIME HR	LRATE kgmol/hr
1	.00000D+00	0.16667D+03
2	.40000D-01	0.16904D+03
3	.54977D-01	0.16724D+03
4	.94977D-01	0.16538D+03
5	.13498D+00	0.16469D+03
6	.13798D+00	0.16464D+03
7	.17798D+00	0.16311D+03
8	.21798D+00	0.16178D+03
9	.25798D+00	0.16070D+03
10	.29798D+00	0.15981D+03
11	.30468D+00	0.15968D+03
12	.34468D+00	0.15867D+03
13	.38468D+00	0.15774D+03
14	.42468D+00	0.15696D+03
15	.46468D+00	0.15628D+03
16	.50468D+00	0.15569D+03
17	.54468D+00	0.15515D+03
18	.58468D+00	0.15466D+03
19	.62468D+00	0.15421D+03
20	.66468D+00	0.15378D+03
21	.70468D+00	0.15336D+03
22	.74468D+00	0.15297D+03
23	.78468D+00	0.15258D+03
24	.82468D+00	0.15220D+03
25	.86468D+00	0.15183D+03
26	.90468D+00	0.15147D+03
27	.94468D+00	0.15110D+03
28	.98468D+00	0.15075D+03

29	.10000D+01	0.15061D+03

* UNIT NO = 5 MODULE NAME = MIXED-TANK *

***

16	.50468D+00	0.35907D+03
17	.54468D+00	0.35845D+03
18	.58468D+00	0.35782D+03
19	.62468D+00	0.35721D+03
20	.66468D+00	0.35663D+03
21	.70468D+00	0.35609D+03
22	.74468D+00	0.35556D+03
23	.78468D+00	0.35506D+03
24	.82468D+00	0.35457D+03
25	.86468D+00	0.35409D+03
26	.90468D+00	0.35363D+03
27	.94468D+00	0.35318D+03
28	.98468D+00	0.35273D+03
29	.10000D+01	0.35256D+03

PROBLEM NO 10

TITLE: PROBLEM NO 10, MIXER-SPLITTER SYSTEM, REF HUSAIN 1986

TOPOLOGY INFORMATION

TOTAL NO OF UNITS	TOTAL NO OF STREAMS
4	4

UNITS DATA

S/N ***	UNIT NAME	UNIT NO	UNIT CODE	UNIT TYPE
1	FEED	1	0	0
2	MIXER	2	21	1
3	SPLITTER	3	9	3
4	SINK	4	0	0

STREAMS DATA

S/N ***	STRM NAME	STRM NO	UNIT FROM	UNIT TO	STRM TYPE
1	S1	1	1	2	3
2	S2	2	2	3	3
3	S3	3	3	4	3
4	S4	4	3	2	3

UNITS CONNECTION MATRIX
****

UNIT NO	UNITS CONNECTED TO IT		
1	2		
2	1	3	3
3	2	4	2
4	3		

STATISTICS OF THE SIMULATION
==========

NO OF FUNCTION EVALUATIONS : 155
NO OF JACOBIAN EVALUATIONS : 12
NO OF STEPS TAKEN : 31
NO OF ERROR TEST FAILURES : 3

RESULTS OF SIMULATION
****

Total number of variables: 7
Total number of equations: 7

---

* UNIT NO = 2 MODULE NAME = MIXER *
****

S/N	TIME HR	LRATE lbmol/hr	XCOMP lbmol/lbmol	XCOMP lbmol/lbmol	XCOMP lbmol/lbmol
1	.00000D+00	0.12500D+03	0.33330D+00	0.33330D+00	0.33340D+00
2	.10000D+00	0.12500D+03	0.32818D+00	0.33362D+00	0.33662D+00
3	.20000D+00	0.12500D+03	0.32337D+00	0.33393D+00	0.33969D+00
4	.30000D+00	0.12500D+03	0.31881D+00	0.33423D+00	0.34263D+00
5	.40000D+00	0.12500D+03	0.31451D+00	0.33451D+00	0.34543D+00
6	.50000D+00	0.12500D+03	0.31048D+00	0.33478D+00	0.34809D+00
7	.60000D+00	0.12500D+03	0.30669D+00	0.33503D+00	0.35062D+00
8	.70000D+00	0.12500D+03	0.30313D+00	0.33528D+00	0.35303D+00
9	.80000D+00	0.12500D+03	0.29979D+00	0.33551D+00	0.35533D+00
10	.90000D+00	0.12500D+03	0.29665D+00	0.33572D+00	0.35751D+00
11	.10000D+01	0.12500D+03	0.29370D+00	0.33593D+00	0.35958D+00
12	.11000D+01	0.12500D+03	0.29093D+00	0.33613D+00	0.36155D+00
13	.12000D+01	0.12500D+03	0.28833D+00	0.33632D+00	0.36342D+00
14	.13000D+01	0.12500D+03	0.28589D+00	0.33650D+00	0.36521D+00
15	.14000D+01	0.12500D+03	0.28359D+00	0.33667D+00	0.36690D+00
16	.15000D+01	0.12500D+03	0.28144D+00	0.33683D+00	0.36852D+00
17	.16000D+01	0.12500D+03	0.27941D+00	0.33699D+00	0.37005D+00
18	.17000D+01	0.12500D+03	0.27751D+00	0.33713D+00	0.37151D+00
19	.18000D+01	0.12500D+03	0.27572D+00	0.33727D+00	0.37290D+00
20	.19000D+01	0.12500D+03	0.27404D+00	0.33741D+00	0.37423D+00
21	.20000D+01	0.12500D+03	0.27246D+00	0.33753D+00	0.37548D+00

APPENDIX B
DASP MINI MANUAL

LIST OF CONTENTS FOR APPENDIX B

		PAGE NO
APPENDIX B1 - MODELLING FOR DASP		199
B1.1	Modelling for Dynamic Simulation	199
B1.1.1	Differential-Algebraic Equations	199
B1.1.2	Integral Equations	200
B1.1.3	Higher Order Differential Equations	200
B1.1.4	Partial Differential Equations	201
B1.2	Modelling for Steady State Simulation	202
APPENDIX B2 - INPUT DATA PREPARATION		203
B2.1	Simulation Using Modules from DASP	204
B2.1.1	CINDAT Data File	204
B2.1.2	CUNIT Data File	207
B2.1.3	CTOPOL Data File	209
B2.1.4	COMDT Data File	212
B2.2	Simulation via "USRSUB" Routine	216
B2.2.1	CINDAT Data File	216
B2.2.2	CUNIT Data File	219
B2.2.3	COMDT Data File	220
APPENDIX B3 - EXECUTION OF DASP		221
B3.1	Introduction	221
B3.2	Starting the Simulation	221
B3.2.1	Using Models from DASP Library	221
B3.2.2	Using User-Written Model Routine	222
B3.3	Calculation of LCMAX and LICMAX	223
B3.3.1	Dynamic Simulation Option	223
B3.3.2	Steady State Simulation Option	225
B3.4	Execution of DASP	226
B3.4.1	Introduction	226
B3.4.2	Initial Region	227
B3.4.3	Dynamic Region	228
B3.4.4	Terminal Region	229

APPENDIX B4 - WRITING A USRSUB ROUTINE 231
B4.1 Introduction 231
B4.2 Description of the Sections 232
B4.2.1 Initialization Section 232
B4.2.2 Function Evaluation Section 233
B4.2.3 Jacobian Evaluation Section 233
B4.2.4 Output Section 234
B4.2.5 Event Code Section 234
B4.2.6 Terminal Section 234

APPENDIX B5 - WRITING A NEW DASP MODULE 236
B5.1 Introduction 236
B5.2 Description of the Argument List 236
B5.3 Description of Common Block Arguments 237
B5.3.1 /MODWK1/ Common Block 237
B5.3.2 /MODWK2/ Common Block 238
B5.3.3 /MODWK4/ Common Block 238
B5.4 Preparation of the Various Sections 238
B5.4.1 Initialization Section 238
B5.4.2 Function Evaluation Section 239
B5.4.3 Jacobian Evaluation Section 240
B5.4.4 Equation and Variable Numbers Setup Section 240

APPENDIX B6 - DASP MODULE LIBRARY 241
B6.1 Controller Module 241
B6.2 Flow Valve Module 243
B6.3 Fractional Valve Opening Module 245
B6.4 Nth Order Module 246
B6.5 Heating Jacket Module 248
B6.6 Cooling Medium Module 251
B6.7 Metal Wall Module 253
B6.8 Variable Volume CSTR Module 255
B6.9 Stream Splitter Module 258
B6.10 Stream Mixer Module 260

APPENDIX B7 - GLOSSARY OF VARIABLE NAMES 263
B7.1 Topology Information 263

B7.1.1	Unit Information	263
B7.1.2	Stream Information	264
B7.1.3	Connection Matrix	264
B7.2	Options Control Variables	265
B7.3	User and Project Information	266
B7.4	Data File Names	269
B7.5	Output Information	269
B7.6	Plotting Information	271
B7.7	Numerical Integration Information	271
B7.8	Algebraic Equation Solver Information	272
B7.9	Event Processing Information	274
B7.10	Variable and Parameter Information	275
B7.10.1	Global Variables	275
B7.10.2	Module Variables and Parameters	276
B7.11	Error Tolerance Information	277
B7.12	Components Information	278

APPENDIX B8 - VARIABLE AND PARAMETER TYPES		281
B8.1	Introduction	281
B8.2	Variable Types	281
B8.3	Parameter Types	283
B8.4	Integer Parameters	285

APPENDIX B9 - ERROR MESSAGES	287

APPENDIX B1

MODELLING FOR DASP

B1.1 MODELLING FOR DYNAMIC SIMULATION

B1.1.1 DIFFERENTIAL-ALGEBRAIC EQUATIONS

DASP can perform the dynamic simulation of any process which can be described by a mixed system of first order differential and linear or nonlinear algebraic equations, known as Differential-Algebraic Equations or DAEs as in equations B1.1 and B1.2 below.

$$y' = f(y, z, t) \qquad \text{..... B1.1}$$

$$0 = g(y, z, t) \qquad \text{..... B1.2}$$

where

- y is the differential variable
- z is the algebraic variable
- y' is the derivative of y
- t is the time
- f is the ordinary differential equation (ODE)
- g is the algebraic equation

Note that more than one derivative term in one equation is permitted. For example, in the modelling of a variable volume CSTR, two of the equations could be the following:

$$\frac{dV}{dt} = F_i - F_o \qquad \text{...B1.2a}$$

$$\frac{d(X_{ok} V)}{dt} = X_{ik}F_i - X_{ok}F_o - r V \qquad \text{...B1.2b}$$

Expanding equation B1.2b, we obtain equation B1.2c

$$V \frac{dX_{ok}}{dt} + X_{ok} \frac{dV}{dt} = X_{ik}F_i - X_{ok}F_o - r\,V \qquad \text{....B1.2c}$$

DASP allows equations to be described as in equation B1.2c, without the need to use some form of approximations to reduce them to the form of equation B1.1. It is possible to transform other forms of equation types into DAEs as described below.

B1.1.2 INTEGRAL EQUATIONS

A system, modelled by the integral equation:

$$I = \int E(t)\,dt \qquad \text{..... B1.3}$$

can be transformed into a differential equation by differentiating the two sides of the equation to obtain

$$\frac{dI}{dt} = E(t) \qquad \text{.....B1.4}$$

B1.1.3 HIGHER ORDER DIFFERENTIAL EQUATIONS

A higher order differential equation can be converted to first order differential equation by introducing m-1 new variables, where m is the order of the equation. For example, a second order differential equation

$$y'' + ky' + y = h(t) \qquad \text{.....B1.5}$$

can be converted into first order by introducing the new variable, $X = y'$, resulting in the two equations

$$X' + kX + y = h(t) \qquad \text{... B1.6}$$

$$y' = X \qquad \text{... B1.7}$$

B1.1.4 PARTIAL DIFFERENTIAL EQUATIONS

Partial differential equations can be converted to first order ordinary differential equations and algebraic equations by the use of the method of lines. The concept of the method of lines can be found in most numerical analysis textbooks and review article by Carver (1981). Consider, for example, the diffusion equation (Davies, 1984).

$$\frac{\delta w}{\delta t} = D \frac{\delta^2 W}{\delta X^2} \qquad \ldots B1.8$$

where $0 < X < 1$, $0 < t$, D = constant

If we choose a mesh on the space interval X so that the appropriate solution will be sought at these mesh:

$$X_1 <, \ldots, < X_{N+1} \qquad \ldots\ldots B1.9$$

For simplicity, let the mesh be of equal length,

$$h = X_{i+1} - X_i, \quad i = 1, N$$

$$X_1 = 0, \; X_{N+1} = 1 \qquad \ldots\ldots B1.10$$

We discretize the spatial derivative in equation B1.8 using finite differences to obtain the following system of ordinary differential equations.

$$\frac{dU_i}{dt} = \frac{D}{h^2} [U_{i+1} - 2U_i + U_{i-1}] \qquad \ldots\ldots B1.11$$

where

$$U_i(t) = W(X_i, t) \qquad \text{.... B1.12}$$

Thus the parabolic partial differential equation has been converted into a coupled system of ODEs in t. The boundary conditions are then converted into subsidiary conditions, which may result in algebraic equations or ODEs.

B1.2 MODELLING FOR STEADY STATE SIMULATION

The equations modelling the steady state behaviour of a plant should be described as a system of linear or nonlinear algebraic equations of the form

$$F(X) = 0$$

where

- X is an N dimensional vector of the unknowns in the equations
- F is the vector of the functions
- N is the number of equations

All other forms of equations must be converted into linear or nonlinear algebraic equations.

APPENDIX B2

INPUT DATA PREPARATION.

B2.1 SIMULATION USING MODULES FROM DASP

This is the case where all the model equations will be assembled from modules in DASP library. The user must prepare appropriate Block units or diagram of the flowsheet of the plant. The following data files must be prepared with the necessary data in them as given in each case:

1. CINDAT, the general input data file,
2. CUNIT, the initial values and error tolerances data file,
3. CTOPOL, the topology data file,
4. COMDT, the components data input data file.

Note that the units (SI or British) of the variables and parameters are given in Appendix 8. Also the data item, blank line, is intended to be used as a comment line as shown in the example data input given below for each data type.

B2.1.1 CINDAT DATA FILE

A. User and problem descriptors

1. Blank line
2. TITLE (up to 60 characters in quotes).
3. CUSER , CPROJ , CDATE.

Example input data is:

Input data for problem number 1 (ref. Luyben 1973).
'Chemical Reactor Simulation using PI-controller'
'J. OGBONDA' , 'EXAMPLE PROBLEM 1' , '10 SEPT 86'

B. Options control variables

1. MODE , KMODEL , LCOMP , KTOL , LPRNT , NONNEG.
2. INDEX , LTERM , JFBS , KJAC , LJAC.

Example input data is:

2 , 0 , 1 , 0 , 1 , 0
1 , 1 , 0 , 1 , .FALSE.

C. Integration parameters.

If MODE = 1 or MODE = 2 then

1. LPLOT , LEVENT , IDISCN , MDEFLT.
2. T0 , TFIN , MFDYN , KDAE.

If MDEFLT = 1 then

3. HMIN , HMAX , H0 , KORDX.

Example input data is:

1 , 1 , 0 , 1
0.0 , 10.0 , 1 , 0
0.0001 , 0.9 , 0.001 , 3

D. Nonlinear equation solver parameters

If MODE = 0 or MODE = 1 then

1. MFALG , KMAX.

Example input data is:

4 , 100

E. Output information

If INDEX = 0 or INDEX = 1 then

1. Blank line
2. NVPRNT.
3. (LUPRNT(j),j=1,NVPRNT).
4. (LSPRNT(j),j=1,NVPRNT).
5. (LVPRNT(j),j=1,NVPRNT).
6. (CPRNT(j),j=1,NVPRNT).

If INDEX = 2 then

1. Blank line
2. NVPRNT.
3. (LUPRNT(j),j=1,NVPRNT) $(1 \leq NVPRNT \leq 10)$

Example input data for INDEX = 1 is:

```
Selected variables to display: 1. Holdup, 2. Control signal, 3. Flowrate.
3
2 , 3 , 4
0 , 0 , 0
42 , 55 , 44
'HOLDUP' , 'SIGNAL-P' , 'FLOWRATE'
```

F. Events information

If LEVENT = 1 and MODE = 1 or 2 then

1. Blank line
2. NEVENT.
3. (KECODE(j),j=1,NEVENT).
4. (EVTIME(j),j=1,NEVENT).
5. (KEUNIT(j),j=1,NEVENT).
6. (KSUNIT(j),j=1,NEVENT).
7. (KVARNO(j),j=1,NEVENT).
8. (TRSVAL(j),j=1,NEVENT).
9. (TRSTOL(j),j=1,NEVENT).
10. (IDCROS(j),j=1,NEVENT).
11. (CENAME(j),j=1,NEVENT).

Example input data is:

```
Information about events.
2
1 , 2
0.0 , 0.0
0 , 2
0 , 0
0 , 42
0.0 , 10.0
0.0 , 0.00001
0 , 1
'TIME EVENT' , 'STATE EVENT'
```

G. Plotting information

If LPLOT = 1 or 2 and MODE = 1 or 2 then

1. Blank line

2. NVPLOT.
3. (LUPLOT(j),j=1,NVPLOT).
4. (LSPLOT(j),j=1,NVPLOT).
5. (LVPLOT(j),j=1,NVPLOT).
6. (CPLOT(j),j=1,NVPLOT).

Example input data is:

Selected variables to plot: 1. Holdup, 2. Control signal, 3. Flowrate.
3
2 , 3 , 4
0 , 0 , 0
42 , 55 , 44
'HOLDUP' , 'SIGNAL-P' , 'FLOWRATE'

B2.1.2 CUNIT DATA FILE

I. INITIAL VALUES

For each unit, taken in the order in which they appear in IUNOS(*) in the topology data file described in Section B2.1.3, the following information will be needed:

If the unit is a module (that is any unit other than INPUT OR OUTPUT unit), then:

1. Integer type parameters, (IEP(j),j=1,MPMI).
2. Real type parameters, (REP(j),j=1,MPMR).
3. Variables of the unit, (X(j),j=1,MVAR).
4. Derivatives of the variables of the unit, (DX(j),j=1,MVAR)

If the unit is an INPUT or OUTPUT module then:

1. Total number of forcing functions(ie. the input or output stream variables in the unit, MPMR.
2. Code numbers of the forcing functions in the unit, (ICEP(j),j=1,MPMR).
3. Initial values of the forcing functions in the unit, (REP(j),j=1,MPMR).

Note that all forcing functions (variables of the INPUT or OUTPUT units) are regarded as parameters and are thus stored in parameter storage locations.

A. MODULES

Each module has different number and type of the variables and parameters and these may depend on the model and parameter options chosen. The user should refer to each module to find out the options available and the parameters and variables needed. Also, the derivatives of the variables are only needed if the integer type parameter IDERIV is equal to 1.

Example input for the controller module:

For a proportional controller (MOPTN=1) in which the setpoint is constant (MPARAM=0) the input signal is liquid holdup (ICODEI=42) and the controller signal is pneumatic (ICODEO=55), with the derivative of the signal not provided (IDERIV=0), the following input data items apply:

1. Blank line
2. MOPTN , MPARAM , ICODEI , ICODEO , IDERIV.
3. ZI , RI , ZO , RO , Y0 , AXN , GAIN , CMAN , SP.
4. PSIGN.

Examaple input data is:

Unit no 10

1 , 0 , 42 , 55 , 0

3.0 , 15.0 , 3.0 , 15.0 , 10.0 , -1.0 , 0.5 , 10.0 , 10.0

10.0

B. INPUT AND OUTPUT UNITS

In general, the values of the following forcing functions or input/output stream variablesshould be provided:

1. Blank line
2. TEMPERATURE (code: 51).
3. PRESSURE (code: 50).
4. FLOWRATE (code: 43 for vapor/gas and 44 for liquid).
5. ENTHALPY (code: 52 for vapor/gas and 53 for liquid).
6. MOLE FRACTIONS (code: 1-20 for vapor, 21-40 for liquids).

Example input data for an input module in which cold water flows into a tank module:

Unit No 20

4

51 , 50 , 44 , 53

300.0 , 1.2 , 100.0 , 1.0E+05

II. ERROR TOLERANCES

This section is only provided for each real module if the user specified the option KTOL=1 or KTOL=2 in the general input data file, CINDAT.
For each **real module**, taken in the order in which they appear in IUNOS(*), the following data items are needed:

1. Blank line
2. (RTOL(j),j=1,MVAR).

If KTOL = 2, then

3. (ATOL(j),j=1,MVAR).

Example input data is:

Unit No 2

0.0001 , 0.00001 , 0.001

0.01 , 0.5 , 0.2

B2.1.3 CTOPOL DATA FILE

The flowsheet of the plant should be translated into the block diagram, which should be made up of units as recognised by DASP (that is they may be UNIT, STAGED or SECTIONALIZED modules or INPUT and OUTPUT units). For example, the flowsheet of a distillation column (fig. B2.1) can be translated into the block module as shown in fig. B2.2. The numbers in brackets are the user supplied integer labels of the units while the stream numbers are shown by each stream. The user supplied names of the units are displayed inside the box representing the units (except the staged unit with name, TOWER). Note that we have chosen in the case of the real units to name them after their module names, although any other names may be given.

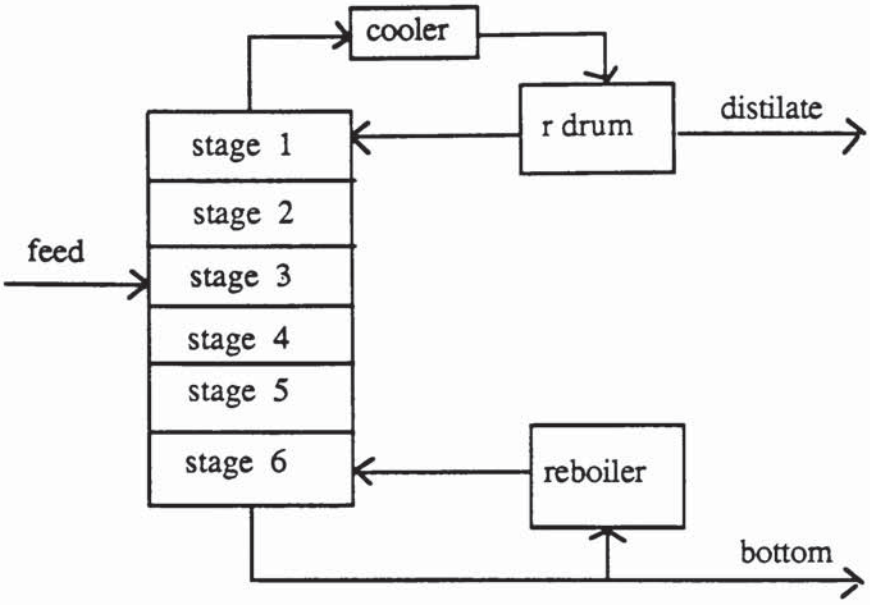


Fig. B2.1 (flowsheet of a distillation column)

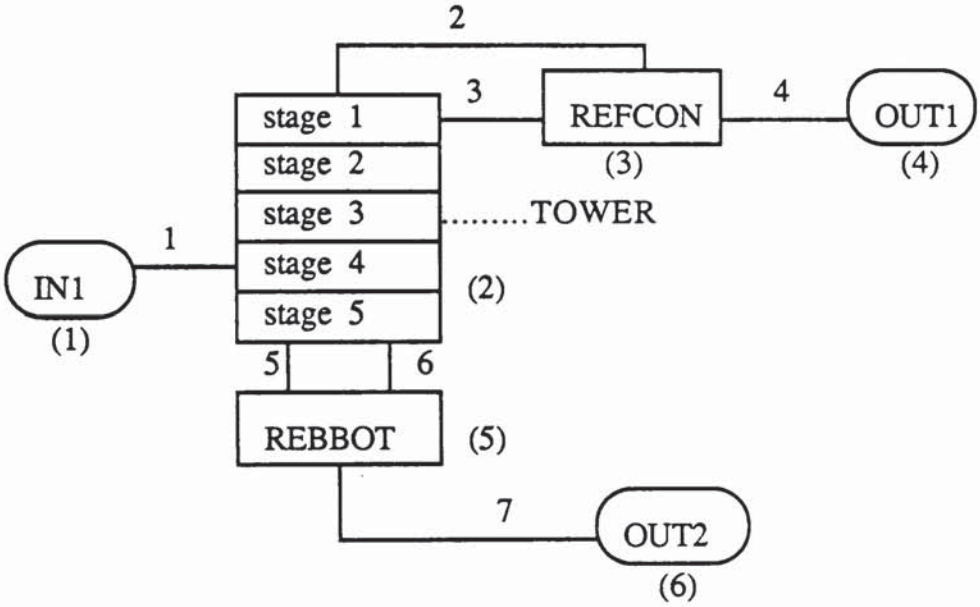


Fig. B2.2 (block module of a distillation column)

The user should provide the following information direct from the block diagram:

A. Space

1. Blank line

B. Units data

1. Blank line
2. NUNIT.
3. (IUNOS(j),j=1,NUNIT).
4. (IUCODE(j),j=1,NUNIT).

5. (IUMOD(j),j=1,NUNIT).
6. (CUNAME(j),j=1,NUNIT).

C. Streams data

1. Blank line
2. NSTRM.
3. (ISNOS(j),j=1,NSTRM).
4. (IUFROM(j),j=1,NSTRM).
5. (IUTO(j),j=1,NSTRM).
6. (ISTYPE(j),j=1,NSTRM).
7. (CSNAME(j),j=1,NSTRM).

D. Connection Matrix of the Units

1. Blank line

For each unit, IU, in the block diagram, taken in the order in which they appear in IUNOS(*), the user should provide the following information:

2. Blank line
3. NS.
4. MP(IU,j),j=1,NS)

For a staged (IUMOD(IU) = 2) or sectionalized (IUMOD(IU) = 3) module, the stage or section number of connection to the other units given in MP(IU,*) should also be provided as:

5. (MC(IU,j),j=1,NS).

Note that those units connected to the unit with index IU, via the primary streams should be given first in the order of the serial numbers of these streams, followed by those connected via secondary streams. This is very important as any wrong connection may be very disastrous

For a unit, the stage or section number of connection apply only if the unit is a staged or sectionalized module, otherwise this data item is not needed.

Example of input data for the distilation column shown above:

Topology of distillation column number 1.

Units information:

```
6
1 , 2 , 3 , 4 , 5 , 6
0 , 10 , 11 , 0 , 12 , 0
0 , 2 , 1 , 0 , 1 , 0
'IN1' , 'TOWER' , 'REFCON' , 'OUT1' , 'REBBOT' , 'OUT2'
```

Streams information:

```
7
1 , 2 , 3 , 4 , 5 , 6 , 7
1 , 2 , 3 , 3 , 5 , 2 , 5
2 , 3 , 2 , 4 , 2 , 5 , 6
3 , 1 , 3 , 3 , 1 , 3 , 3
'S1' , 'S2' , 'S3' , 'S4' , 'S5' , 'S6' , 'S7'
```

Connection Matrix for the distillation problem

```
IN1
    1
    2
TOWER
    5
    1 , 3 , 3 , 5 , 5
    4 , 1 , 1 , 5 , 5
REFCON
    3
    2 , 4 , 2
OUT1
    1
    3
REBBOT
    3
    2 , 2 , 6
OUT2
    1
    5
```

B2.1.4 COMDT DATA FILE

For the calculation of K-values and/or enthalpies of the components, the user can choose between three options:

1. UNIQUAC model.
2. UNIFAC model.
3. POLYNOMIAL model.

Alternatively, the user can provide his own model via KVAL04 or ENTH04 interface routine for K-values or enthalpy calculation respectively.

I. FOR ALL CASES

In all cases the user must supply the values of the following constants/parameters:

1. Blank line
2. Blank line
3. NCX , MFK , MFH , IDEAL.
4. (IDX(j),j=1,NCX).

If the number of components in each unit is not the same (for the case of KMODEL=0 or KMODEL=1), then the user should provide the number and identities of components in the modules (if LCOMP=2).

5. For each real unit, taken in the order in which they appear in IUNOS(*), the following data items should be given:
   i. Blank line
   ii. NC.
   iii. (ID(j),j=1,NC)

6. (DMWT(j),j=1,NCX).

II. FOR THE UNIQUAC OR UNIFAC MODEL

1. Blank line
2. (TC(j),j=1,NCX).
3. (PC(j),j=1,NCX).
4. (DM(j),j=1,NCX).
5. (RD(j),j=1,NCX).
6. (ZR(j),j=1,NCX).

III. FOR UNIQUAC METHOD ONLY

1. Blank line
2. (RU(j),j=1,NCX).

3. (QU(j),j=1,NCX).
4. (QP(j),j=1,NCX).

5. Provide solvation and association parameters, ETA(*), in the following format:
   ETA(1,1)
   ETA(2,1) , ETA(2,2)
   : : : : : : : : : : : :
   ETA(NCX,1) , ETA(NCX,2) , ::::::, ETA(NCX,NCX)

6. For i=1,NCX (ie NCX data on each line),
   U(i,j),j=1,NCX)

7. For i=1,NCX (ie 5 data on each line),
   (C1(j),C2(j),C3(j),C4(j),C5(j),j=1,NCX).

If enthalpy calculation using the UNIQUAC model is needed, then provide also the following data items:

8. For i=1,NCX (ie 4 data on each line),
   (D1(j),D2(j),D3(j),D4(j),j=1,NCX).

IV. FOR UNIFAC METHOD ONLY

1. Blank line
2. Provide solvation and association parameters, ETA(*), in the following format:
   ETA(1,1)
   ETA(2,1) , ETA(2,2)
   : : : : : : : : : : :
   ETA(NCX,1) , ETA(NCX,2) , ::::::, ETA(NCX,NCX)
3. For i=1,NCX (5 data on each line)
   (C1(j),C2(j),C3(j),C4(j),C5(j),j=1,NCX).

4. NG.
5. (IDGP(j),j=1,NG).
6. For i=1,NCX
   i. NG1
   ii. (ITAB(i,j),j=1,NG1)
   iii. (NY(i,j),j=1,NG1)

If enthalpy calculation using the UNIQUAC model within the UNIFAC model is needed, then provide also the following data items:

7. For i=1,NCX (ie 4 data on each line),
   (D1(j),D2(j),D3(j),D4(j),j=1,NCX).

V. FOR POLYNOMIAL METHOD ONLY

1. Blank line
2. For i=1,NCX (ie 5 data on each line),
   (C1(j),C2(j),C3(j),C4(j),C5(j),j=1,NCX).

3. For i=1,NCX (ie 5 data on each line),
   CV1(j),CV2(j),CV3(j),CV4(j),CV5(j),j=1,NCX)

4. For i=1,NCX (ie 5 data on each line),
   (CL1(j),CL2(j),CL3(j),CL4(j),CL5(j),j=1,NCX)

B2.2 SIMULATION VIA 'USRSUB' ROUTINE

This is the case where all the model equations will be supplied by the user via the USRSUB interface routine. The user must write the USRSUB routine as described in Appendix B4 of this manual. The following data files must be prepared with the necessary data in them as described in each case:

1. CINDAT, the general input data file,
2. CUNIT, the initial values and error tolerances data file,
3. COMDT, the components data input data file.

The types of data and the order of providing them in the data files are described below. Also the data item, blank line, is intended to be used as a comment line as shown in the example data input given below for each data type.

B2.2.1 CINDAT DATA FILE

A. User and problem descriptors

1. Blank line
2. TITLE (up to 60 characters in quotes).
3. CUSER , CPROJ , CDATE.

Example input data is:

Input data for problem number 1 (ref. Luyben 1973).
'Chemical Reactor Simulation using PI-controller'
'J. OGBONDA' , 'EXAMPLE PROBLEM 1' , '10 SEPT 86'

B. Options control variables

1. MODE , KMODEL , LCOMP , KTOL , LPRNT , NONNEG.
2. INDEX , LTERM , JFBS , KJAC , LJAC.

Example input data is:

2 , 2 , 1 , 0 , 1 , 0
1 , 1 , 0 , 1 , .FALSE.

C. Integration parameters

If MODE = 1 or MODE = 2 then

1. LPLOT , LEVENT , IDISCN , MDEFLT.
2. T0 , TFIN , MFDYN , KDAE.

If MDEFLT = 1 then

3. HMIN , HMAX , H0 , KORDX.

Example input data is:

1 , 1 , 0 , 1
0.0 , 10.0 , 1 , 0
0.0001 , 0.9 , 0.001 , 3

D. Nonlinear equation solver parameters

If MODE = 0 or MODE = 1 then

1. MFALG , KMAX.

Example input data is:

4 , 100

E. Output information

If INDEX = 0 or INDEX = 1 then

1. Blank line
2. NVPRNT (Max. 10)
3. (LVPRNT(j),j=1,NVPRNT).
4. (CPRNT(j),j=1,NVPRNT).
5. (CUPRNT(j),j=1,NVPRNT).

If INDEX = 2 then

1. Blank line
2. NVPRNT (1≤NVPRNT≤NVAR)
3. (ICX(j),j=1,NVPRNT)

Example input data is:

Selected variables to display: 1. Holdup, 2. Control signal, 3. Flowrate.
3

1 , 2 , 3
'HOLDUP' , 'SIGNAL-P' , 'FLOWRATE'
'KG-MOL' , 'PSIG' , 'KG/HR'

F. Events information

If LEVENT = 1 then

1. Blank line
2. NEVENT (Max. 10)
3. (KECODE(j),j=1,NEVENT).
4. (EVTIME(j),j=1,NEVENT).
5. (KVARNO(j),j=1,NEVENT).
6. (TRSVAL(j),j=1,NEVENT).
7. (TRSTOL(j),j=1,NEVENT).
8. (IDCROS(j),j=1,NEVENT).
9. (CENAME(j),j=1,NEVENT).

Example input is:

Information about events.
2
1 , 2
0.0 , 0.0
1 , 2
0.0 , 10.0
0.0 , 0.00001
0 , 1
'TIME EVENT' , 'STATE EVENT'

G. Plotting information

If LPLOT = 1 or 2 and MODE = 1 or 2 then

1. Blank line
2. NVPLOT. (Max 5)
3. (LVPLOT(j),j=1,NVPLOT).
4. (CPLOT(j),j=1,NVPLOT).

Example input data is:

Selected variables to plot: 1. Holdup, 2. Control signal, 3. Flowrate.
3
1 , 2 , 3

'HOLDUP' , 'SIGNAL-P' , 'FLOWRATE'

H. Sparse Jacobian matrix information

If JFBS = 1 then

1. Blank line
2. NZ
3. (IEQN(j),j=1,NZ).
4. (JVAR(j),j=1,NZ).

Example input data is:

Structure of Jacobian marix.
10
1,1,1,2,2,3,3,3,4,4
1,2,4,2,3,1,3,4,1,2

B2.2.2 CUNIT DATA FILE

A. INITIAL VALUES

The user must supply the following information:

1. Blank line
2. Blank line
3. NVAR , NEQN , IDERIV.
4. Blank line
5. (XVAR(j),j=1,NVAR).

If IDERIV = 1 then

6. Blank line
7. (DXVAR(j),j=1,NVAR)

If MODE = 1 or 2

8. Blank line
9. (EDYN(j),j=1,NVAR)

If NONNEG = 2

10. Blank line

11. (IDNEG(j),j=1,NVAR)

Example input for the controller module

Initial values for proportional control of a CSTR.
Initial values
3 , 3 , 1
variable values
10.0 , 10.0 , 100.0
derivatives
-1.0 , 0.009 . 0.11
EDYN(*) values
1.0 , 0.0 , 0.0

II. ERROR TOLERANCES

If KTOL = 1 or KTOL = 2 then

The user must supply the following information:

1. Blank line
2. Blank line
3. (RTOL(j),j=1,NVAR)

If KTOL = 2 then

4. Blank line
5. (ATOL(j),j=1,NVAR)

Example input data is:
1=holdup, 2=controller signal, 3=flowrate
relative error toleraces
0.00001 , 0.00001 , 0.0001
absolute error toleraces
0.1 , 0.1 , 1.0

B2.2.4 COMDT DATA FILE

All data must be supplied as described in Section B2.1.4, with the exception that item *I/5* is not needed.

APPENDIX B3
EXECUTION OF DASP

B3.1 INTRODUCTION

In order to run the simulator, the user must decide which of the following options he wants to use to described his problem to DASP:

1 Using DASP modules (DM)
2 Using a combination of user module in DASP format and DASP library modules (NDM)
3 Using only user module via USRSUB interface routine (UM).

In the first case, the user will ask DASP to assemble the equations describing his problem from the library of model routines. These are made up of unit operations or parts thereof. In the second case, some of the modules describing the user's problem are not available in DASP module library. Thus the user will write the missing subroutines as described in Appendix B5, which are then compiled and linked with the DASP package to produce the executable file, DASP. In the third case, the user will provide the model equations describing his problem via an interface routine, USRSUB, which is described in Appendix B4. In this case these subroutines are compiled and linked to the DASP package producing an executable (or output) file, DASP.

Whichever option is chosen, the user must then prepare the data files, which contain his problem description, initial values of variables and parameters in his problem as described in Appendix B2. Finally, the package is run by typing

"DASP" < ENTER>

Note anything the user has to type on the terminal is put in double quotes as above and <ENTER> is equivalent to depressing the ENTER key on the keyboard.
This general procedure described here is valid for the IBM PC/AT at the Chemical Engineering Dept using RM Fortran 77 compiler.

B3.2 STARTING THE SIMULATION

B3.2.1 USING MODELS FROM DASP LIBRARY

The following files in C:/JOEL/MDASP/ directory are needed if you want to run it on another directory or disk.

1 PDASP.BAT
2 PDASP1.EXE
3 MDASP.BAT
4 MDASP.LNK
5 MESFLE.D

1 If all these files are available, then prepare the input data files as explained in Appendix B2 (ie. CINDAT, CUNIT and CTOPO2 data file and COMDT if chemical species are present in the system).

2 Type "PDASP" < ENTER >

3 The user is requested to enter the name of the topology data file which has already been prepared and the maximum lengths of the real and integer work spaces, CORE(*) and ICORE(*) arrays used by DASP. The calculation of these lengths are described in section B3.3. Using the topology data file, PDASP routine prepares two routines, main program, DASP and subroutine GETSUB in the file, MDASP.FOR if the topology file was correctly prepared. The subroutines in this file are then compiled to produce MDASP.OBJ as the object file.

4 Type "MDASP" <ENTER>.
The MDASP.OBJ file is linked with all the DASP programs and libraries resulting in the output or executable file, DASP.EXE.

5 Type "DASP" <ENTER>
to start execution of DASP.

B3.2.2 USING USER-WRITTEN MODEL ROUTINE

The following files in C:/JOEL/UDASP/ directory are needed if the user wants to run DASP in another directory or disk.

1 PDASP.BAT
2 PDASP1.EXE
3 UDASP.BAT
4 UDASP.LNK
5 MESFLE.D

1 Prepare the subroutine USRSUB in the file, UDASP.FOR as described in Appendix B4. This file must also contain the following main program

```
PROGRAM DASP
PARAMETER (LCMAX = ..., LICMAX = ...)
DOUBLE PRECISION CORE (LCMAX)
INTEGER ICORE (LICMAX), IFLAG
EXTERNAL USRSUB
IFLAG = 0
CALL DASPM (USRSUB, CORE, LCMAX, ICORE, LICMAX, IFLAG)
STOP
END
```

This program should then be followed by the user routine, "USRSUB". Note the name, "USRSUB" is a variable name. It can be called any other name. The length of LCMAX and LICMAX should be calculated as described in Section B3.3. Example LICMAX=500, LCMAX = 1000. Note also that the name of the file, UDASP.FOR can be regarded as a variable name provided the user edits the file, UDASP.LNK and changes the name, UDASP to the new file name.

2 Compile the file, UDASP.FOR using the RM Fortran 77 Compiler command

"RMFORT UDASP"

If the subroutines are correctly written, there will be no errors and UDASP.OBJ will be generated as the object file.

3 Type "UDASP" <ENTER>

This will link UDASP.OBJ with DASP library routines, generating the output or executable file DASP.EXE

4 To run the program type "DASP" <ENTER>

B3.3 CALCULATION OF LCMAX and LICMAX

B3.3.1 DYNAMIC SIMULATION OPTION

The lengths of the work arrays, CORE and ICORE can be calculated as given below. For the floating point work array, CORE, the minimum length, LCMAX is:

$$LCMAX = L1 + L2 + L3 + L4 + L5 + L6 + L7 + L8 + L9 + L10 + L11$$

For the interger work array, ICORE, the minimum length, LICMAX is

$$LICMAX = I1 + I2 + I3 + I4 + I5 + I6 + I7 + I8 + I9 + I10 + I11$$

The Ls and Is are explained below:

(a) Workspace needed by DASP Executive

L1 = 2 * NVAR

I1 = NVAR

(b) Workspace needed for storing the values of variables and derivatives

L2 = 2 * NVAR

I2 = 0

(c) Workspace needed for storing the code numbers of variables

I3 = NVAR

L3 = 0

(d) Workspace for storing the values of parameters

If KMODEL = 0 or 1

I4 = MPI

L4 = MPM

If KMODEL = 2

I4 = 0

L4 = 0

(e) Workspace needed for storing code numbers of parameters

If KMODEL = 0 or 1

I5 = MPM

L5 = 0

If KMODEL = 2

I5 = 0

L5 = 0

(f) Workspace needed for storing variable types (algebraic or differential)

I6 = 0

L6 = NVAR

(g) Workspace needed for storing error tolerances

I7 = 0

If KTOL = 0

L7 = 2

If KTOL = 1 or 2

L7 = 2 * NVAR

(h) Workspace needed for storing equation and variable numbers

L8 = 0

If JFBS = 0

I8 = 0

If JFBS = 2

I8 = 2 * NZ

(i) Workspace needed by DASSL integrator (MFDYN = 1)

L9 = 40 + 9 * NVAR

I9 = 35

If JFBS = 0

L10 = NVAR * NVAR

I10 = NVAR

If JFBS = 2

L10 = 3 * NZ + NVAR

I10 = 5 * NZ + 13 * NVAR

(j) Workspace needed for calculating VLE and enthalpy of components

If no VLE and enthalpy calculation using UNIQUAC or UNIFAC is required, then

I11 = 0

L11 = 0

For the calculation of VLE and enthalpy using UNIQUAC or UNIFAC, then:

For using the UNIQUAC routines

I11= 0

L11 = 12 * (NCX + 1) * NCX/2 + NCX * (NCX + 8)

For using the UNIFAC routines,

I11 = MAX (IKK - IK, 0)

L11 = MAX (LKK - LK, 0) + 12 * NCX + 1) * NCX/2 + NCX (NCX+ 8)

where

IKK = 76

IK = I10 (as given above)

LKK = 1752

LK = L9 + L10 (as given above)

B3.3.2 STEADY STATE SIMULATION OPTION

The same as in dynamic simulation described in section B3.3.1, except:

L6 = 0

I6 = 0

For Broyden's method (MFLAG = 2)

L9 = NVAR + NVAR * 14 + 24

I9 = 0

For NRSUB routine (MFLAG = 4)

L9 = NVAR * (NVAR + 5)

I9 = NVAR

B3.4 EXECUTION OF DASP

B3.4.1 INTRODUCTION

In order to run the DASP program, the user should do the following:

1 Prepare the necessary data files as described in Appendix B2.

2 For the UM option, the user must write the subroutine, USRSUB and the problem routines as described in Appendix B4. These routines must be compiled and linked with DASP library programs to produce the executable file, DASP.EXE.

3 For the NDM option, the user must code the new routines in DASP format as described in Appendix B5. These routines must then be compiled and linked with DASP library programs to produce the executable file, DASP.EXE.

The package is run by typing

"DASP" < ENTER >

The system displays the following:

Welcome to DASP

(Dynamic Analysis and Simulation Package)

Version 1.1 1986, 1987

** Initialization Region **

Press < ENTER > to continue ...

The last message was produced by a routine that clears the screen if the < ENTER> key

is depressed on the keyboard. A similar routine clears the screen automatically. The first option gives the user the opportunity to view the displayed message before it disappears.

B3.4.2 INITIAL REGION

The following message will appears:

```
** Enter units used for input data **
1 = SI Units          2 = British Units
-------------------------------------------
```

There are only two types of units supported, and the user can only use either of the two to input his data to DASP (see Appendix B8 for details of these units). However, if UM option is used, then it does not matter which of these is used as the user can describe the problem consistently in any units.

The user is prompted to enter the names of the data files and the result files. Note that if any typing errors are made, the user has up to 3 trials after which execution is stopped. When any input data file name is read, it is opened for sequential access and the values in the file are read. If any errors are detected in the input data files, the execution is terminated. If scalar relative error tolerance option is to be used (KTOL = 0) then the user is prompted to enter the scalar error tolerance. If any discontinuity will occur in the model at a specified time, TSTOP, then the user is prompted for the value of TSTOP. If the user wants output to be given at specified output intervals, then the value of this interval is prompted. If the user wants output at specified time interval (in dynamic simulation option) or after a number of iterations (in steady state simulation option), then he is prompted for the value of HOUT, the output time interval or NOUT, the number of iterations before output.

Afterwards, the following is displayed:

```
** Enter interrupt/break flag (LINTRP) as follows **
        0 = No break/interrupt
        1 = Break after a given number of steps/iterations
        2 = Break at every step/iteration
       -1 = Interrupt/stop simulation now.
```

If LINTRP = 0, then integration will proceed from initial time, T0 to the final time, TFIN without any break, except when an event has occurred. If LINTRP = 1, then the user wants control returned to him via the MODIFY routine (explained in Chapter 8) after a given number of steps/iterations. In this case he is prompted for the value of

NINTRP, the number of iterations before break. For LINTRP = 2, this is equivalent to a break after every iteration or step. However, if LINTRP = -1 is entered, the simulation is abandoned and a message asking the user whether to stop or restart the simulation is displayed.

The plotting routine is called to initialize the plotting parameters, if plotting option is specified. Note that if the UM option is being executed, then the initial section of the subroutine USRSUB (ie. JS = 0) is called to do any initializations specified by the user. To end the initialization process, the consistency of the values of the simulation variables are verified and if any inconsistency is detected, simulation is abandoned. Finally, the MODIFY routine is called. On exit from the MODIFY routine, simulation enters the dynamic region.

B3.4.3 DYNAMIC REGION

This region carries out the numerical calculations of the steady state and dynamic simulation sessions. At the end of every step or iteration, control is passed to the Executive program to check which flags have been set. First it checks if any error has occurred (ie. if IFLAG < 0), in which case the relevant error message is displayed and control is passed to the error analysis section of the Terminal Region. It then checks if the end of simulation has been reached (LEND=TRUE), in which case control is passed to the Rerun/End simulation section of the Terminal Region. It then checks if the event flag, LEVENT and/or break flag, LINTRP has been set in which case the event processing interface routine, STESUB or the MODIFY routine is called to carry out the necessary procedure. Note that the user can abandon the simulation during this time by setting LINTRP to a negative value using option 10 of the MODIFY routine.

If none of these flags has been set, but the user has requested for output at specified time intervals or after a certain number of iterations, then the routine, OUTPUT is called to write the values of variables and derivatives to the work file, WKFILE and to the screen if LTERM=1. Finally, on return from any of the above routines, a check is made to find out if any errors were detected, in which case the relevant error message is displayed and control passed to the error analysis section of the Terminal Region. Otherwise, if the user has set LINTRP to a negative value, signifying an interrupt, then the simulation is abandoned and the message

SIMULATION ABANDONED BY USER AT THE TIME .....
Select an option as follows:
0 = STOP
1 = RESTART

If 0 is selected, then all open files are closed after the results have been written to the results file and the simulation is abandoned. If 1 is selected, then the results are written to the results file, all open files closed, the pointers reset to zero and the simulation is restarted.

If no interupts or errors were found, then the relevant simulation interface routine (DERIV for dynamic and ASSUB for steady state) is called to continue the simulation. This procedure is repeated until the final time, TFIN is reached and control is passed to the Terminal Region.

B3.4.4 TERMINAL REGION

At the terminal region, if any errors have occurred typified by IFLAG < 0, then using the absolute value of the flag, the relevant error messages are displayed on the screen and further messages from the messages file, MESFLE and execution is stopped. Otherwise, the following message appears:

```
****** TERMINAL REGION *****
        Terminal Menu is
  0 = Exit/End simulation
  1 = Rerun present system after perturbation
  2 = Run a new system with data in same files
  3 = Run a new system using new data files
  4 = Modify, plot or view variables/parameters
     ** Select an option number **
```

To exit, you must select option 0 above, in which case, the results are written to the result file and all files closed before stopping the simulation by returning control to the main program, accompanied by the display of the message

NORMAL END OF SIMULATION

To view the latest results or plot the values of variables against time (if any plotting routines are available) select option 4, which calls the MODIFY routine. To rerun present system after a call to the MODIFY routine, select option 1. The user can restart the simulation starting from any step between the initial values and the latest variable values. Choosing option 2 means that the user wants to start a new simulation of a problem whose data are in the same files, placed just after the data for a previous simulation. In this case the result of the previous simulation is written to the result file, and an initialization of a new run will be started.

In option 3, the present simulation is ended and a reinitialization activated for a new run. Since all previous data files are closed, a new run must use new data files.

Depending on the option chosen, after a return from the Terminal Region, the Executive directs the simulation to the relevant region.

APPENDIX B4
WRITING A USRSUB ROUTINE

B4.1 INTRODUCTION

A user who wants to use his own model equations instead of DASP module library can do so by preparing the Fortran subroutine, USRSUB as shown below:

```
C*****************************************************************
      SUBROUTINE USRSUB(LIN,LOUT,JS,TIME,XVAR,DXVAR,FUNC,
     *                  PD,NVAR,NEQN,CJ,KFLAG,RPAR,IPAR)
C-----------------------------------------------------------------------------------------
C ARGUMENTS ARE:
C   LIN: Logical device number for input.
C   LOUT: Logical device number for output.
C   JS  : Section control variable:
C         = 0 Initialization section,
C         = 1 Function evaluation,
C         = 2 Jacobian evaluation,
C         = 3 Output section,
C         = 4 Event processing section,
C         = 5 Terminal section,
C   TIME : Current time.
C   XVAR(*) : Current values of variables in the module.
C   DXVAR(*) : Current values of the derivatives of XVAR(*).
C   FUNC(*) : Vector to hold the residuals of the equations.
C   PD(*) : Vector to hold the Jacobian matrix values.
C   NVAR: The total number of variables in the model.
C   NEQN: The total number of equations in the model.
C   CJ  : Real type variable passed to this routine by the integrator,
C         for Jacobian evaluation.
C   KFLAG : Integer flag, set to negative value if error occured.
C   RPAR(*): Dummy real array, may be used in this routine.
C   IPAR(*): Dummy integer array, may be used in the routine.
C-----------------------------------------------------------------------------------------
C
      KFLAG = 0
      IF(JS .EQ. 0)THEN
C        **Do all necessary initializations ***
      ELSEIF(JS .EQ. 1)THEN
C        **Calcualte the function values***
      ELSEIF(JS .EQ. 2)THEN
C        ***Calculate the Jacobian values ***
      ELSEIF(JS .EQ. 3)THEN
C        ***OUTPUT PARAMETERS AND VARIABLES ***
      ELSEIF(JS .EQ. 4)THEN
C        ***EVENT CODE SECTION ******
      ELSEIF(JS .EQ. 5)THEN
C        ***TERMINAL SECTION ******
      END IF
      RETURN
      END
```

The structure of this routine is similar to that of a DASP module, with divisions of initial, function evaluation, Jacobian evaluation, output, event code and terminal sections typified by the value of JS as 0, 1, 2, 3, 4 and 5 respectively.

Comments:

1. All the sections above except for JS=1 are optional and may be neglected.
2. The logical device numbers LIN and LOUT are for input and output from and to the default output except for JS=5, when all output will be directed to the result file given by the user.

B4.2 DESCRIPTION OF THE SECTIONS

All the sections will be described using the example problem:

$$dX_1/dT = 2*X_1 + X_1*X_2$$
$$dX_2/dT = 5*X_2 + Y^2*X_1 \qquad \ldots 1$$
$$0 = EPS*Y - 15*X_2$$

The variables are: X_1, X_2, Y. The parameter is EPS.

B4.2.1 INITIALIZATION SECTION

This section is called by DASP during the Initial Region. The user can do all necessary initializations during this time such as reading the values of parameters and forcing functions. Also the values of derivatives could be read or calculated if not provided in the initial values data file. Also other forms of calculations can be done such as steady state simulation by calling a program written by the user for that purpose.

For the example problem (equation (1) above), the setup could be like this:

```
      SUBROUTINE USRSUB(LIN,LOUT,JS,TIME,XVAR,DXVAR,FUNC,
     *                  PD,NVAR,NEQN,CJ,KFLAG,RPAR,IPAR)
C-----------------------------------------------------------------------------
      DOUBLE PRECISION TIME, XVAR(*), DXVAR(*), FUNC(*),
     *                 PD(*), CJ, RPAR(*)
      INTEGER LIN, LOUT, JS, NVAR, NEQN, KFLAG, IPAR(*)
C
      KFLAG = 0
      IF(JS .EQ. 0) THEN
         WRITE(LOUT, ERR=999,FMT=20)
20       FORMAT(10X, 'ENTER THE VALUE OF EPS')
         READ(LIN, ERR=999,FMT=*)EPS
         RPAR(1) = EPS
         RETURN
999      KFLAG = -7
         RETURN
```

Comment:
RPAR and IPAR can be used to store the values of parameters and control variables. They are not modified by DASP and have been dimensioned RPAR(50) and IPAR(50) in a common block in DASP.

B4.2.2 FUNCTION EVALUATION SECTION

Here the residuals of the equations are calculated. The user can either program the equations above in a seperate subroutine (problem routine) and call it here or program the equations here. The former is useful in including subroutine of procedures in which the output variables are calculated instead of the residuals. The equations must be written in the form:

$$FUNC(1) = F_1\ (T,Y,Y')$$

$$FUNC(2) = F_2\ (T,Y,Y')$$

$$\vdots \qquad \vdots$$

$$FUNC(NEQN) = F_{NEQN}\ (T,Y,Y')$$

For the example equations, this section could look like this:

```
      ELSEIF(JS .EQ. 1)THEN
         EPS = RPAR(1)
         X1 = XVAR(1)
         X2 = XVAR(2)
         DX1 = DXVAR(1)
         DX2 = DXVAR(2)
         Y = XVAR(3)
C
         FUNC(1) = 2.0*X1 + X1*X2 - DX1
         FUNC(2) = 5.0*X2 + Y*Y*X1 - DX2
         FUNC(3) = EPS*Y - 15.0*X2
C-------------------------------------------------------------------------------------
```

Thus DASP forwards the current values of TIME, XVAR(*) and DXVAR(*) to the subroutine so that the residulas of the equations will be calculated and returned in FUNC(*). If for any reasons the residuals could not be calculated, KFLAG should be set to a negative value.

B4.2.3 JACOBIAN EVALUATION SECTION

If analytical Jacobian matrix of the equations are available, or if the user wants to use a different method of calculating the Jacobian instead of the finite difference method used in DASP, then this section should be used for that purpose. For dense Jacobian matrix, the elements should in forwarded via FUNC(*) vector arranged column by column as in

$$df_1/dx_1,\ df_2/dx_1,\ ....,\ df_1/dx_2,\ df_2/dx_2\ ,...,\ df_1/dx_3,\ ..$$

For the sparse option (JFBS=2), the elements should be forwarded in the order of supplying the equation (IEQN(*)) and variable (JVAR(*)) numbers of the nonzeros given in the general input data file, CINDAT.

For the example problem above using dense Jacobian matrix option:

```
      ELSEIF(JS .EQ. 2)THEN
         EPS = RPAR(1)
         X1 = XVAR(1)
         X2 = XVAR(2)
         DX1 = DXVAR(1)
         DX2 = DXVAR(2)
         Y = XVAR(3)
C
         PD(1) = 2.0 + X2 - CJ*DX1
         PD(2) = Y
         PD(3) = 0.0
         PD(4) = X1
         PD(5) = 5.0 -CJ*DX2
         PD(6) = -15.0
         PD(7) = 0.0
         PD(8) = X1
         PD(9) = EPS

C-------------------------------------------------------------------------------------------
```

B4.2.4 OUTPUT SECTION

This section can be used to output the values of variables and parameters at the output times to the terminal. However, if the user wants to format the output in a different form without using the DASP output format (INDEX=4), LOUT will be the unit number for the result file at this section so that all output will be written to the result file.

B4.2.5 EVENT CODE SECTION

The user can provide the event code for his problem in this section. For example, an event may involve changing the value of a variable or parameter and so on. However, if the equations are changed, then KFLAG should be set to a positive nonzero integer value to indicate to the integrator to restart the integration. Also the number of variables and equations must be modified accordingly.

B4.2.6 TERMINAL SECTION

This section is call by DASP at the end of the simulation to execute those statements

which are meant to be executed at this time.

Finally, the Fortran 77 program should be completed by the following statements:

```
END IF
RETURN
END
```

This program should be compiled and loaded with DASP library program to produce an executable output file DASP.

APPENDIX B5

WRITING A NEW DASP MODULE

B5.1 INTRODUCTION

A DASP module is divided into seven sections according to the value of JS. A typical DASP module is of the following form shown in figure B5.1.

```
C*************************************************************************
      SUBROUTINE MODULE( IC, JS, T, X, DX, ICX, F, IEP, REP,
     *                   ICEP, CJ, IRES, RP, IP,  CR, ICR )
C-------------------------------------------------------------------------------------------
      INTEGER  IC, JS, ICX(*), IEP(*), ICEP(*), IRES, IP, ICR(*)
      DOUBLE PRECISION  T, X(*), DX(*), F(*), CJ, RP, CR(*)
      COMMON /MODWK1/ IU,LDNIN,LDNOUT,MVAR,MFUNC,MPMI,MPMR,
     *                IUOPT,IUNC,IOPTN,NC,ID(20),ICVR(50),IPM(20)
      COMMON /MODWK2/ FX(100),RPM(20),YF(20),XF(20),HV(20),HL(20)
      COMMON /MODWK4/ DFX(100)
C-------------------------------------------------------------------------------------------
      IRES = 0
      IF(JS .EQ. 0)THEN
C        --------Initial Section--------
      ELSEIF(JS .EQ. 1)THEN
C        --------Function Evaluation Section---
      ELSEIF(JS .EQ. 2)THEN
C        ---------Jacobian Evaluation Section----
      ELSEIF(JS .EQ. -1)THEN
C        -------Forwarding of Sparse Jacobian Elements Section----
      ELSEIF(JS .EQ. 3)THEN
C        --------Output Section-------
       ELSEIF(JS .EQ. 4)THEN
C        ----------Event Code Section---
       ELSEIF(JS .EQ. 5)THEN
C        ----------Terminal Section---
      END IF
      RETURN
      END
```

Figure B5.1 Structure of a DASP Module

B5.2 DESCRIPTION OF THE ARGUMENT LIST

IC Code number of the module routine in DASP library.

JS Section control variable:

= 0 Initialization section,

= 1 Function evaluation,

= 2 Jacobian evaluation,

= 3 Output time,

= 4 Event processing section,

= 5 Terminal section,

= -1 Forward equation and variable numbers of Jacobian matrix.

T Current time.

X(*) Current values of variables in the module.

DX(*) Current values of the derivatives of X(*).

ICX(*) Code numbers of the variables.

F(*) Vector to hold the residuals of the equations.

IEP(*) Integer type equipment parameters in the module.

REP(*) Real type equipment parameters in the unit.

ICEP(*) Code numbers of the real type parameters.

CJ Real type variable passed to modules by the integrator for Jacobian evaluation.

IRES Integer flag, set to negative value if error occured.

RP Dummy real variable, may be used in the module.

IP Dummy integer variable, may be used in the module.

CR(*) Real storage space. Must be declared as DOUBLE PRECISION CR(*). Must not be changed or used as workspace.

ICR(*) Integer storage space. Must not be changed or used as workspace. Must be declared as : INTEGER ICR(*)

B5.3 DESCRIPTION OF COMMON BLOCK ARGUMENTS

B5.3.1 /MODWK1/ COMMON BLOCK

IU Index of the label of this unit in IUNOS(*).

LDNIN logical device number for input.

LDNOUT logical device number for output.

MVAR number of variables in the unit, set according to the value of the model option.

MFUNC number of equations in the unit, set according to the value of the model option.

MPMI number of integer type parameters in the unit, set according to the value of the model/parameter option.

MPMR number of real type parameters in the unit, set according to the value of the model/parameter option.

IUOPT The units used by the use to input the variables and parameters of the model. The values are as follows:
= 1 for SI Units
= 2 for British Units.

IUNC unit type. On entering the module, it comes with the unit number of the module, as given by the user in the topology information file, CTOPOL. During function (JS=1) or Jacobian (JS=2) evaluations, it is passed on to this routine as the user-supplied label of the unit

whose variables are being retrieved. If this value is negative, then that unit has been removed from the flowsheet.

IOPTN analysis type option. It gives the type of analysis to be done as follows:
= 1 for steady state simulation
= 2 for dynamic simulation
= 3 for design
= 4 for optimization.

NC the number of components in this unit.

ID(*) the identity of the components in the unit.

ICVR(*) integer work array, used to transfer the code numbers of variables and parameters.

IPM(*) integer work array.

B5.3.2 /MODWK2/ COMMON BLOCK

FX(*) real work array, used to transfer the values of variables.

RPM(*),YF(*),XF(*),HV(*),HL(*) are real work arrays.

B5.3.3 /MODWK4/ COMMON BLOCK

DFX(*) real work array, used to transfer the values of derivatives of variables being retrieved.

B5.4 PREPARATION OF THE VARIOUS SECTIONS

B5.4.1. INITIALIZATION SECTION

1. Set the number of integer type parameters (MPMI).
2. Set the code numbers of the parameters in ICVR(*) array.
3. Read the integer type parameters into IEP(*) by calling:

```
CALL UNIPMI(LDNIN,IEP,ICVR,MPMI,IRES)
```

4. Check if error occured (ie. IRES < 0) and take appropriate action.
5. Set the number of real type parameters (MPMR) according to the values of model/parameter options.
6. Set the code numbers of the real type parameters (ICEP(*))
7. Read the real type parameters into REP(*) by calling:

```
CALL UNIPMR(LDNIN,IEP,ICVR,MPMI,IRES)
```

8. Check if error occured (ie. IRES < 0) and take appropriate action.
9. Set the number of variables (MVAR) according to model/parameter options.
10. Set the code numbers of the variables in ICX(*)
11. Read the values of the variables into X(*) (and their derivatives into DX(*) if IDERIV=1) by calling:

```
CALL UNIVAR(LDNIN,IDERIV,X,DX,MVAR,IRES)
```

12. Check if error occured (ie. IRES < 0) and take appropriate action.
13. Return to the calling subprogram.

B5.4.2. FUNCTION EVALUATION SECTION

In this section the residuals of the model equations are calculated. The procedure is as follows:

1. Retrieve the values of the input variables and parameters, in the order of the serial numbers of the connection streams. This is done in the following order for each stream connected to this module:
   i. Set the serial number (ISNO).
   ii. Call the retrieve routine via:

```
CALL RTRV(ISNO,IRES,CR,ICR)
```

   iii. Check if error occured and take appropriate action.
   iv. Check the value of IUNC to find out if the unit has been removed from the flowsheet, in which case the stream/unit variables should be equated to zero. IUNC has the following values:
      < 0 if the unit has been removed from the flowsheet,
      = the unit label otherwise.
   v. Transfer the values of the input stream variables to temporary variables and/or arrays from FX(*) according to their code numbers.
   vi. Transfer the values of the input stream derivatives to temporary variables and/or arrays from DFX(*) according to their code numbers.
   vii. Go to (i) above if more input stream variables are to be retrieved.

2. Calculate the residuals of the equations, which should be written in the form:

$$F(1) = F_1(T,X,DX$$

$$\vdots \qquad \vdots$$

$$F(MFUNC) = F_{MFUNC}(T,X,DX)$$

B5.4.3. JACOBIAN EVALUATION SECTION

In this section the partial derivatives of the model equations with respect to the variables are calculated. The procedure is as follows:

1. As in function evaluation section.
2. Calculate the Jacobian matrix. The partial derivatives of the functions are calculated column by column for the dense option, that is:

$$df_1/dx_1,\ df_2/dx_1,\ ....,\ df_1/dx_2,\ df_2/dx_2\ ,....,\ df_1/dx_3,\ ..$$

   For the sparse option, the values are calculated in the order in which the equation and variables numbers of the nonzeros were forwarded in JS=-1 section.

These values should be returned in F(*) array.

B5.4.4. EQUATION & VARIABLE NUMBERS SETUP SECTION

In this section the equation and variable numbers of each variable in each equations is forwarded by calling the routine SETJAC as:

CALL SETJAC(IEQ,ISNO,MVR,IRES,IND,CR,ICR)

where

- IEQ The equation number, taken serially.
- ISNO The serial number of the connecting stream for the case of input nonzeros. For variables of this unit ISNO=0.
- MVR The number of nonzeros in a unit. Note that the nonzeros of this unit are referenced first, then those of input units one by one for each particular equation.
- IND Integer variable with values as follows:
  = 0 if SETJAC is called for nonzeros of this unit.
  > 0 for the input or ouput units.

The order is as follows:

For each equation, with IEQ = equation number,

i. Forward the variable numbers of this unit with IND=0, ISNO=0
ii. Forward the variable numbers of each input unit with IND > 0 and ISNO=primary stream number of the input stream (as given in Connection block of the module concerned).

Note that in each call of SETJAC routine, the code numbers of the nonzeros concerned should be setup in ICVR(*).

The sections JS=3, JS=4 and JS=5 are the same as decribed in Appendix 4.

APPENDIX B6
DASP MODULE LIBRARY

This section presents the modules available in DASP Library. In each case, the module is described with respect to what it is intended to do (description), the modelling equations, the connection block which gives the streams connected to it and the order in which the user is expected to give the units connected to this module via these streams. Also presented are the subroutine name, module type and module code. The parameters and variables of the module are described, and depending on the value of the integer type parameters chosen, the model, the parameters and variables needed may be different for each option within a module.

B6.1 CONTROLLER MODULE

B6.1.1 DESCRIPTION

This routine describes the model of proportional (P), proportional-integral (PI) and a proportional-integral-derivative (PID) controllers.

B6.1.2 MODEL EQUATIONS

P-Controller

$$Y = Y0 + GAIN * (ERR)$$

PI-Controller

$$Y = Y0 + GAIN * \left(ERR + \frac{1}{TI} \int ERR\, dT\right)$$

PID-Controller

$$Y = Y0 + GAIN * \left(ERR + \frac{1}{TI} \int ERR.dT + TD \frac{dERR}{dT}\right)$$

where

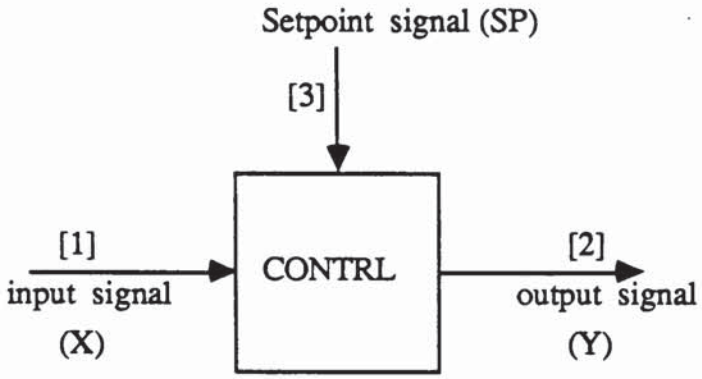
ERR = AXN * (SP - X)

Y is the output signal

X is the input signal

If the controller is in manual mode, then the output signal is set to a constant value, CMAN.

B6.1.3 CONNECTION BLOCK



B6.1.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
CONTRL	1	1

B6.1.5 INTEGER PARAMETERS

The following integer type parameters should be supplied in the order given:

1	MOPTN	2	MPARAM
3	ICODE	4	ICODEO
5	JOPTN	6	IDERIV

In the above list,

MOTPN = 1 for P - controller
= 2 for PI-controller
= 3 for PID-controller

MPARAM = 0 for constant setpoint
= 1 for variable setpoint, ie. the setpoint is input from another unit.

ICODEI is code number of input signal, X

ICODEO is code number of output signal, Y

JOPTN = 0 for automatic mode
= 1 for manual mode.

B6.1.6 REAL PARAMETERS

1	ZI	2	RI
3	ZO	4	RO
5	Y0	6	AXN
7	GAIN	8	CMAN

If MOPTN = 1

9 SP (if MPARAM = 0)

If MOTPN = 2

9 TI

10 SP (if MPARAM = 0)

If MOPTN = 3

9 TI

10 TD

11 SP (if MPARAM = 0)

Comment: The units of SP is the same of that of X, the input signal.

B6.1.7 VARIABLES

1 Y

Note: Y = PSIGN or ESIGN depending on the code number, ICODEO

B6.1.8 DERIVATIVES OF VARIABLES

1 dY if IDERIV = 1

B6.2 FLOW VALVE MODULE

B6.2.1 DESCRIPTION

Routine models a valve, through which vapor/gas or liquid flows.

B6.2.2 MODEL EQUATIONS

Vapor

$F = AV * CV * CONST * P1$ if $P2 < 0.53 * P1$

$F = AV * CV * CONST * \sqrt{PP (P1 - P2)}$ otherwise

where $PP = (P1 + P2)/2$

Liquid

$F = AV * CONST * CV * \sqrt{(P1 - P2)}$

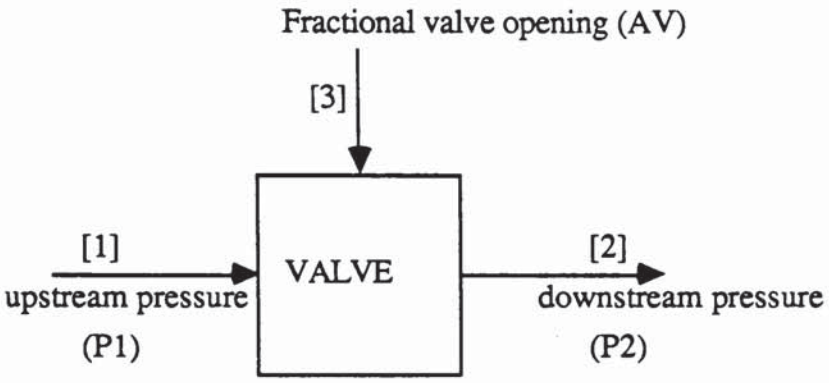
where

P1 is the upstream pressure

P2 is the downstream pressure

F is the flowrate through the valve

B6.2.3 CONNECTION BLOCK



B6.2.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
VALVFL	2	4

B6.2.5 INTEGER PARAMETERS

1 MOPTN

2 MPARAM

3 IDERIV

where

MOPTN = 1 for fixed position valve

= 2 for control valve

MPARAM = 1 for vapor or gas flow

= 2 for liqiud flow

B6.2.6 REAL PARAMETERS

1 CV

2 CONST

3 AV

In the above CONST is a constant factor to multiply with the valve constant, CV to make the units consistant.

B6.2.7 VARIABLES

1 F (where F is VRATE or LRATE).

B6.2.8 DERIVATIVES

1 dF (if IDERIV = 1)

B6.3 FRACTIONAL VALVE OPENING MODULE

B6.3.1 DESCRIPTION

Module models the relationship between fractional valve opening, AV, and valve stem position, using either linear, square root or equal percentage characteristics.

B6.3.2 MODEL EQUATIONS

The relationships between valve stem position, PSP and fractional valve opening can be any of the following:

Linear

AV = AV0 + (1 - AV0) PSP

Square Root

AV = AV0 + (1 - AV0) * SQRT (PSP)

Equal Percentage

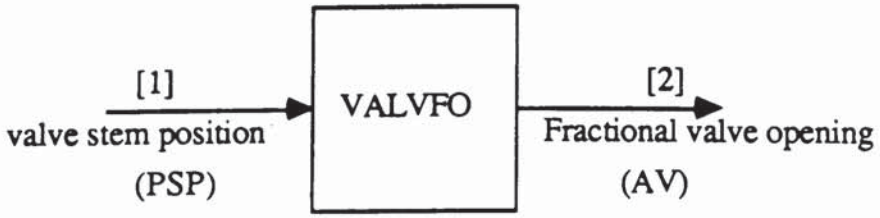
AV = AV0 * EXP (PSP/AVV)

where

AVV = -100.0/LOG (AV0)

AV0 is the valve opening with fully closed stem position

B6.3.3 CONNECTION BLOCK



B6.3.4 MODULE DESCRPTORS

Module Name	Module Code	Module Type
VALVFO	3	1

B6.3.5 INTEGER PARAMETERS

1 MOPTN
2 ICODEI
3 ICODEO
4 IDERIV

where

MOPTN = 1 for linear
= 2 square root
= 3 equal percentage

ICODEI: Code number of PSP

ICODEO: Code number of AV

B6.3.6 REAL PARAMETERS

1 ZI zero of valve stem position
2 RI range of valve stem position
3 ZO zero of valve opening
4 RO range of valve opening
5 AV0 where $0 \leq AV0 \leq 1$

B6.3.7 VARIABLES

1 AV

B6.3.8 DERIVATIVES

1 dAV (If IDERIV = 1)

B6.4 Nth ORDER MODULE

B6.4.1 DESCRIPTION

This routine models a zero (or simple gain), first or second order system. These systems include valves, valve stem position, sensors, transmitters etc which can be described by any of these functions:

B6.4.2 MODEL EQUATIONS

Zero order

$Z1 = Y0 + (X\,1 - X0) * GAIN$

First order

$$\tau \frac{dZ1}{dt} + Z1 = GAIN * X1$$

Second order

$$\tau^2 \frac{d^2Z1}{dt} + 2.\varepsilon.\tau \frac{dZ1}{dt} + Z1 = GAIN * X1$$

where

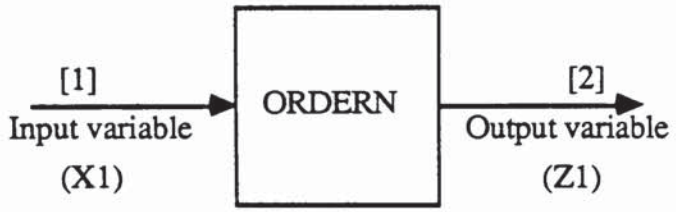
X1 is the input variable/signal

Z1 the output variable/signal

τ time constant

ε damping ratio

B6.4.3 CONNECTION BLOCK



B6.4.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
ORDERN	4	1

B6.4.5 INTEGER PARAMETERS

1	MOPTN	2	ICODEI
3	ICODEO	4	IDERIV

where

MOPTION = 0 for zero order

= 1 for first order

= 2 for second order

ICODEI: Code number of input variable

ICODEO: Code number of ouput variable

B6.4.6 REAL VARIABLES

1	ZI	2	RI
3	ZO	4	RO
5	X0	6	Y0

If MOPTN = 0

7	GAIN		

If MOPTN = 1

7	TAU	8	GAIN

If MOPTN = 2

7	TAU	8	ZETA (= ε)
9	GAIN		

B.6.4.7 VARIABLES

1	Z1

B6.4.8 DERIVATIVES

1	dZ1	(If IDERIV = 1)

B6.5 HEATING JACKET MODULE

B6.5.1 DESCRIPTION

This module models a heating jacket, which can be attached to a tank. The heating medium is assumed to be vapor (eg. steam).

B6.5.2 MODELLING EQUATIONS

TS and PS Constant

$$QS = HS * AS * (TS - TM)$$

$$\rho S \frac{dVS}{dt} = WS - QS/\lambda S$$

PS and VS Constant

$$QS = UHTC * AS * (TM - TS)$$

$$\rho S * VS * CPS \frac{dTS}{dt} = WS * CPS * (TSO - TS) + QS$$

where

$\rho S = MS * PS/(R * TS)$ (the vapor density)

MS = molecular wt of vapor/medium

AS = steam/vapor side surface area

HS = vapor side heat transfer coefficient

λS = latent heat of vaporization

PS = pressure in the jacket

TS = temperature in the jacket

QS = heat transfer rate

VS = volume of vapor in jacket

UHTC is the overall heat transfer coefficient

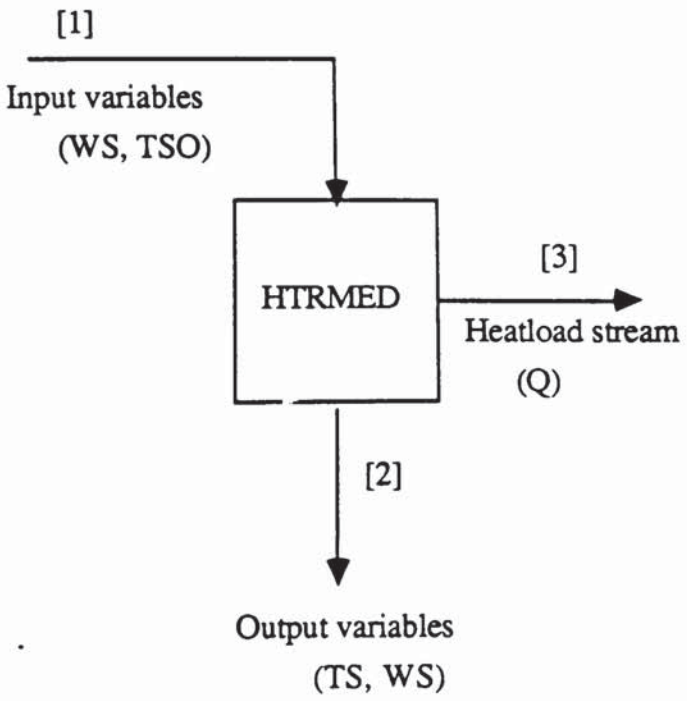
CPS is the heat capacity of vapor

TSO is the input stream temperature

WS is the input steam flowrate

TM is the temperature of the process side

B6.5.3 CONNECTION BLOCK



B6.5.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
HTRMED	5	1

B6.5.5 INTEGER PARAMETERS

1 MOPTN 2 IDERIV

where

MOPTN = 1 constant TS and PS

= 2 constant PS and VS

B6.5.6 REAL PARAMETERS

If MOPTN = 1

1 HS 2 AS

3 λS 4 MS

5 TS 6 PS

If MOPTN = 2

1 PS 2 VS

3 CPS 4 UHTC

5 AS 6 PS

7 ρS

B6.5.7 VARIABLES

If MOPTN = 1

1 QS 2 VS

If MOPTN = 2

1 QS 2 TS

B6.5.8 DERIVATIVES

If IDERIV = 1, then

For MOPTN = 1

1 dQS 2 dVS

For MOPTN = 2

1 dQS 2 dTS

B6.6 COOLING MEDIUM MODULE

B6.6.1 DESCRIPTION

This routine models a cooling jacket, which can be attached to any tank or reactor, eg. CSTR, to cool the contents.

B6.6.2 MODEL EQUATIONS

The model equations depend on the model option being chosen:

Cooling at constant volume in jacket

$$QW = HW * AW * (TM - TW)$$

$$\rho W * VW * CW * \frac{dTW}{dt} = WI * CW * TI - WO * CW * TW + QW$$

Cooling at varying volume

$$QW = HW * AW * (TM - TW)$$

$$\rho W*VW*CW*\frac{dTW}{dt} = WI* CW*TI - WO*CW*TW + QW - \rho W * CW *TW*\frac{dVW}{dt}$$

$$\rho W * \frac{dVW}{dt} = WI - WO$$

where

CW is the heat capacity of fluid

ρW is the density of fluid

HW is the medium side heat transfer coefficient

AW is the medium side surface area

VW is the volume of medium fluid in the jacket

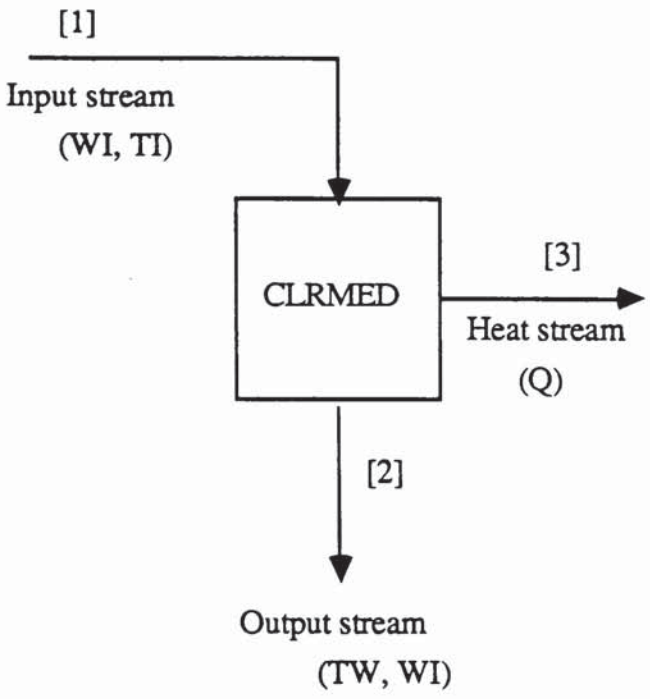
TW is the jacket temperature

WI is the inlet flow rate

WO is the outlet flow rate

QW is the heat load

B6.6.3 CONNECTION BLOCK



B6.6.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
CLRMED	6	1

B6.6.5 INTEGER PARAMETERS

The following integer type parameters are needed:

1 MOPTN
2 IDERIV

where

MOPTN = 1 for constant volume
= 2 for variable volume

B6.6.6 REAL PARAMETERS

Depending on the model option, MOPTN, chosen, the following parameters are needed:

MOPTN = 1

1 CP
2 LDENS
3 HIO
4 AREA
5 LVOL

MOPTN = 2

1	CP	2	LDENS
3	HIO	4	AREA

B6.6.7 VARIABLES

The variables depend on the model option, MOPTN, selected

MOPTN = 1

1	HTLOAD	2	TEMP

MOPTN = 2

1	HTLOAD	2	TEMP
3	LVOL		

B6.6.8 DERIVATIVES

The derivatives of the variables described in Section B6.6.7 above

B6.7 METAL WALL MODULE

B6.7.1 DESCRIPTORS

This routine models the metal wall between a process and the heating or cooling medium in a jacket.

B6.7.2 MODEL EQUATIONS

The model equations depend on the model options as follows:

Lumped metal wall model

$$QM = HM * AM * (TM - TP)$$

$$\rho M * CM * VM * \frac{dTM}{dt} = Q1 - QM$$

where

CM is the heat capacity of the wall

ρM is the density of wall

VM is the volume of the wall

HM is the wall heat transfer coefficient

AM is the surface area

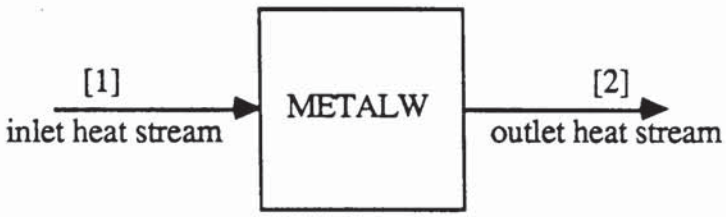
TM is the wall temperature

Q1 is the heat load from either the heating or cooling medium

QM is the heat load from the wall to the process side

TP is the process side temperature

B6.7.3 CONNECTION BLOCK



B6.7.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
METALW	7	1

B6.7.5 INTEGER PARAMETERS

1 MOPTN 2 IDERIV

where

MOPTN = 1 for lumped wall model.

B6.7.6 REAL PARAMETERS

For MOPTN = 1

1 LDENS 2 CP

3 LVOL 4 HIO

5 AREA

B6.7.7 VARAIBLES

For MOPTN = 1

1 HTLOAD 2 TEMP

B6.7.8 DERIVATIVES

The derivatives of the variables in Section B6.7.7 above.

B6.8 VARIABLE VOLUME CSTR MODULE

B6.8.1 DESCRIPTION

Routine models a variable volume CSTR (ie. Continuous Stirred Tank Reactor), in which a number of reactions, NR ($1 \leq NR \leq 20$) are taking place with any order of reaction. In order to use this routine, the user is expected to write a routine, REACTR, which will be called by this module to calculate the rates and the heats of reactions.

B6.8.2 MODEL EQUATIONS

The modelling equations can be expressed as follows:

$$\frac{\rho R}{MR} * \left(VR * \frac{dX_i}{dt} + X_i \frac{dVR}{dt}\right) = (FI * XI_i - FO * X_i)/MR + RATEI_i$$

$$\rho R \frac{dVR}{dt} = FI - FO$$

$$FO = CV * CONST * \sqrt{\rho R} * VR$$

$$PO = P + 9.81 * VR * \rho R/AT$$

If MOPTN = 2

$$CP * \rho R * \left(\frac{dTR}{dt} * VR + TR \frac{dVR}{dt}\right) = CP * (FI * TI - FO * TO) + \sum_{i=1}^{NR} RATE0_j \, \Delta HR_j + Q$$

where

ρR is the density of the reaction fluid

VR is the volume of the reaction fluid

TR is the reaction temperature

FI is the total inlet flowrate

FO is the outlet flowrate

TO is the outlet/reactor temperature

X_i is the reactor mole fractions for component i

MR is the average reactor mixture molecular weight

AT the area of the reaction mixture

XI_i the mole fraction for component i in the inlet stream

ΔHR_j is the heat of reaction for reaction j

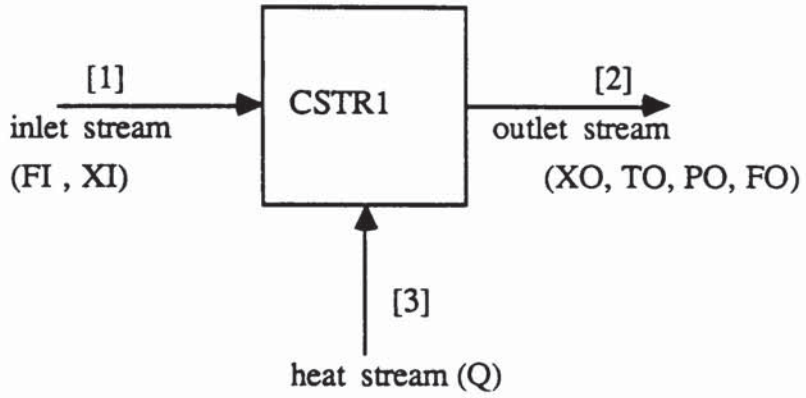
$RATE0_j$ is the combined rate of reaction of reaction j

$RATEI_i$ is the rate of reaction of component i

Q is the heat load to or from the reactor

In the above CONST is a constant factor to multiply with the valve constant, CV to make the units consistant.

B6.8.3 CONNECTION BLOCK



B6.8.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
CSTR1	8	1

B6.8.5 INTEGER PARAMETERS

1	MOPTN	2	MPARAM
3	NR	4	IDERIV

where

MOPTN = 1 for isothermal

= 2 for nonisothermal

MPARAM = 0 for constant heat input (Q = constant)

= 1 for variable heat input (Q is input from another unit)

NR is the number of reaction schemes

B6.8.6 REAL PARAMETERS

These depend on the model option chosen:

MOPTN = 1

1	LDENS	2	AREA
3	CV	4	PRES
5	TEMP		

MOPTN = 2

1	LDENS	2	AREA
3	CV	4	PRES
5	CP	6	HTLOAD

B6.8.7 VARIABLES

The variables are as follows:

1	XCOMP (i), i = 1, NC	2	LVOL
3	LRATE	4	PRES

If MOPTN = 2

5 TEMP

B6.8.8 DERIVATIVES

The derivatives of the variables mentioned in Section B6.8.7 above.

B6.8.9 STRUCTURE OF A REACTR ROUTINE

To use the CSTR1 routine, the user must write a routine, REACTR, which should have the following format:

```
SUBROUTINE  REACTR  (IUN,KEY,X,ID,NC,NR,V,RHO,T,P,
*                           RATEI,RATEO,HTR, IFLAG)
DOUBLE PRECISION X (*), V, RHO, T, P, RATEI (*), RATEO (*), HTR (*)
INTEGER IUN, KEY, ID (*), NC, NR, IFLAG
        |
        |   Calculate reaction rates, RATEI(*), RATEO(*)
        |   Calculate the heats of reaction, HTR(*)
        |
RETURN
END
```

where

IUN is the unit number of the module calling this routine
KEY is the section control variable
= 0 for initial
= 1 for function evaluation
= 2 for Jacobian
= 3 for output
= 4 for event section
= 5 for terminal section
X(*) is a vector of mole fractions of components
ID(*) is a vector of identities of components
NC is the number of components

NR is the number of reactions
V is the volume of reaction mixture
RHO is the density of the mixture
T is the temperature
P is the pressure
RATEI(*) is a vector of components reaction rates
RATEO(*) is a vector of overall reaction rate of each reaction
HTR(*) is a vector of heats of reaction
IFLAG is a performance flag, which should be set to a negative value if any errors occur.

Note that it is possible to use this same subroutine to calculate these variable values (ie. RATEI, RATEO, and HTR) for other CSTR1's if it appears more than once in the flowsheet. This is done using the Fortran 77 IF-ELSEIF-ENDIF blocks and the unit number, IUN.

B6.9 STREAM SPLITTER MODULE

B6.9.1 DESCRIPTION

The routine models a stream splitter, in which one input stream can be split into two or more (up to 20) output streams. The split fractions can all be constant or varying.

B6.9.2 MODEL EQUATIONS

The general equation is given as:

$F_i = \alpha_i FI, \quad i = 1, NOS$

where

F_i is the i^{th} output stream flowrate

FI is the input stream flowrate

α_i is the i^{th} output stream split fraction

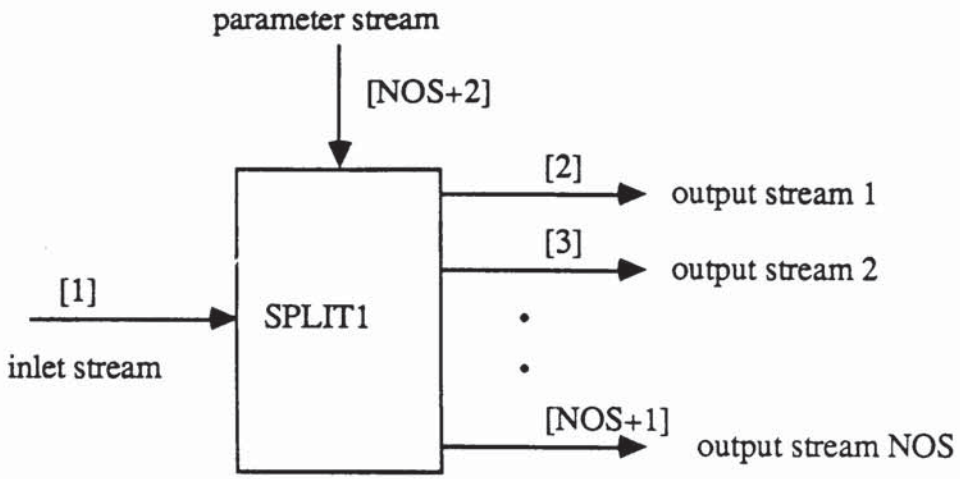
NOS is the number of output streams

If the α's are variable, then it is assumed that the α of the first output stream, α_1^{new} is input from another module. This will then be used to convert the old α's to their respective new values as follows:

$$a1 = (1 - \alpha_1^{new})/(1 - \alpha_1^{old})$$

$$\alpha_1 = a1 * \alpha_i^{old}, \qquad \text{for } i = 2, NOS$$

B6.9.3 CONNECTION BLOCK



Note: Stream [NOS + 2] above is only needed if the α's are variable.

B6.9.4 MODULE DESCRIPTORS

Module Name	Module Core	Module Type
SPLIT1	9	3

B6.9.5 INTEGER PARAMETERS

1 NOS
2 MPARAM
3 ICODEI
4 IDERIV

where

NOS is the number of output streams

MPARAM is parameter option variable

= 0 for constant split fractions, α's

=1 for variable split fractions

ICODEI is the code number of split fraction signal, which is input to the first output stream (if MPARAM = 1).

B6.9.6 REAL PARAMETERS

1 $RATIO_i$, i=1, NOS

If MPARAM = 1, then

2 ZI

3 RI

where

RATIO's are equivalent to the α's

ZI is the zero of input split fraction signal, α_1^{new}

RI is the range of input split fraction signal, α_1^{new}

Note that ZI and RI are used to calculate the new α_1 (MPARAM = 1) as follows:

$$\alpha_1^{new} = \frac{SI}{(RI - ZI)}$$

where

SI is the actual input signal from another input module to output stream No 1.

B6.9.7 VARIABLES

1 $LRATE_i$, for i=1, NOS

B6.9.8 DERIVATIVES

The derivatives of the variables in Section B6.9.7 above.

B6.10 STREAM MIXER MODULE

B6.10.1 DESCRIPTION

This routine models a stream mixer in which NIS input streams ($1 \leq NIS \leq 8$) are combined into 1 output stream with no hold-up in the mixer.

B6.10.2 MODEL EQUATIONS

$$FO = \sum_{i=1}^{NIS} FI_i$$

If there are chemical species in the system (NC > 0), then

$$XO_k = \frac{\sum_{i=1}^{NIS} FI_i \; XI_{ik}}{FO}$$

for k = 1, NC

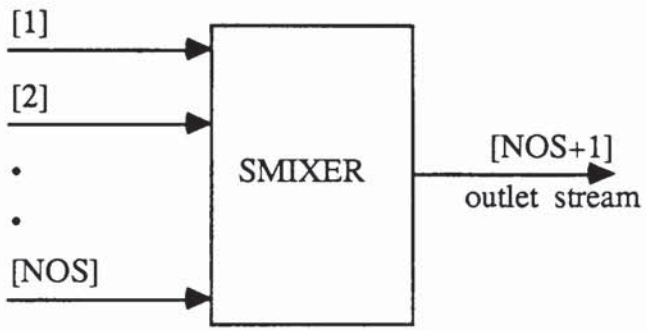
If the temperature is not constant and no change of phase then

$$TO = \frac{\sum_{i=1}^{NIS} FI_i \; TI_i * CI_i}{FO*CO}$$

where

FO is the flowrate of output stream
TO is the temperature of output stream
XO_k is the mole fraction of the k^{th} component in the output stream
XI_{ik} is the mole fraction of the k^{th} component in the i^{th} input stream
FI_i is the i^{th} input stream flowrate
TI_i is the temperature of the i^{th} input stream
NIS is the number of input streams ($1 \leq NIS \leq 8$)
NC is the number of components
CI_i is the specific heat of the ith inlet stream
CO is the specific heat of the outlet stream

B6.10.3 CONNECTION BLOCK



B6.10.4 MODULE DESCRIPTORS

Module Name	Module Code	Module Type
SMIXER	10	1

B6.10.5 INTEGER PARAMETERS

1 MOPTN 2 NIS

3 IDERIV

where

MOPTN = 1 for isothermal, TO = constant

= 2 for nonisothermal

NIS is the number of input streams ($1 < NIS \leq 8$)

B6.10.6 REAL PARAMETERS

1 C_{pi} , i = 1, NC.

B6.10.7 VARIABLES

1 LRATE

If NC > 0, ie. if there are chemical species in the system then

2 $XCOMP_i$, i=1, NC

If MOPTN = 2 then

3 TEMP

B6.10.8 DERIVATIVES

The derivatives of the variables in section B6.10.7 above.

APPENDIX B7
GLOSSARY OF VARIABLE NAMES

B7.1 TOPOLOGY INFORMATION

B7.1.1 UNIT INFORMATION

NUNIT

The total number of modules/units in the block diagram. Note that all input and output streams are regarded as coming from and going to units. They are designated as INPUT (or source) and OUTPUT (or sink) units respectively and are included in NUNIT.

IUNOS(*)

User-supplied positive integer labels of the units in the block diagram. Integer array of length, NUNIT.

IUCODE(*)

The code numbers of the units in the block diagram, as given in DASP modules library, Appendix B6. It is used by the Executive routine to identify the appropriate Fortran code. Note all INPUT and OUTPUT units are given code number zero (0). Integer array of length, NUNIT.

IUMOD(*)

This is the module type as given in the particular module (see Appendix B6). The values are as follows:

0 Any unit which is not a module. For example source or sink units.
1 Unit module.
2 Staged module.
3 Sectionalized module.
4 Valve, Pump, Compressor module.

CUNAME(*)

User-supplied character names of the units in the block diagram. Character array (max. 12 characters per name) of length, NUNIT enclosed in quotes. Note that the units are taken in the order in which they are given in IUNOS(*) above.

B7.1.2 STREAM INFORMATION

NSTRM

The total number of streams in the block diagram. Note the definition of streams here include all forms of information flows such as MATERIAL, ENERGY, SIGNAL, HEAT, etc.

ISNOS(*)

User-supplied positive integer labels of all streams in the block diagram. Integer array of length, NSTRM.

IUFROM(*)

Unit numbers of the units from which a given stream originates. Integer array of length, NSTRM.

IUTO(*)

Unit numbers of the units to which a given stream is going. Integer array of length, NSTRM.

ISTYPE(*)

The phase of the stream, given according to the following convention:

0 All forms of information flow streams, which is neither material nor material nor energy flow. Example is the controller signal flow and heat flow stream.

1 Vapor or Gas flow stream.

2 Mixed Vapor-Liquid flow stream.

3 LIQUID flow stream.

4 Mixed LIQUID/SOLID flow stream.

5 SOLID flow stream

CSNAME(*)

User-supplied character names of the streams in the block diagram. Character array (max. 5 characters per name) of length, NSTRM enclosed in quotes. Note that the streams are taken in the order in which they are given in ISNOS(*) above.

B7.1.3 CONNECTION MATRIX

For each unit with index, IU (taken in the order in which they appear in IUNOS(*)), the following information establishes the connectivity of the units to one another:

NS

The total number of units connected to the selected unit, IU. This includes all units connected via the primary and secondary streams.

MP(IU,*)

The unit numbers of the connected units given in the order of the serial numbers of primary streams followed by those connected via secondary streams. Integer array of length, NS.

MC(IU,*)

For Staged or Sectionalized modules (IUMOD(IU) is equal to 2 or 3), the stage or section number of the module, IU, to which the units given in MP(IU,*) are connected. Integer array of length, NS.

B7.2 OPTIONS CONTROL VARIABLES

JINPUT

Input mode option. This gives the type of input mode to be used as follows:

0 Keyboard input only. Here all data are entered from the keyboard. This option is not available.

1 Keyboard and prepared files input. In this case the values of all the option variables are entered via the keyboard while all detailed input are in prepared files. This is the available option .

2 File input only. This is the situation where all input data is entered via prepared files only. This option is not available.

MODE

Mode of simulation as follows:

0 Steady state simulation. The user must provide the starting initial guesses for all the variables.

1 Steady state and dynamic simulation. The steady state values provide the initial values for dynamic simulation. The user must provide all the initial quesses.

2 Dynamic simulation given initial values. In this case the user is expected to provide all the initial values of the variables.

KMODEL

Source of model routine option with values as follows:

0 All model equations should be assembled from DASP module library routines.

1 Some of the model equations should be assembled from DASP library module routines. The user provides the rest model equations in user-supplied module routines, coded in DASP module format.

2 All model equations are provided by the user via USRSUB routine. This is equivalent to the fully equation-oriented approach.

KTOL

Error tolerances option. This specifies the type of error tolerances to be used as follows:

0 Scalar relative error tolerance, TOL. In this case the user is prompted to enter the error tolerance.

1 Vector relative error tolerances, RTOL(*). These should given by the user in the CUNIT data file.

2 Vector relative (RTOL(*)) and absolute (ATOL(*) error tolerances. These are given by the user in the CUNIT data file.

NONNEG

Nonnegativity option. This specifies whether or not the system should invoke nonnegativity criteria on all variables during numerical calculations. Values are as follows:

0 Do not invoke nonnegativity criteria on all variables. In this case some of the variables may have negative values.

1 Invoke nonnegativity criteria on all variables. In this case all the variables definitely have positive values.

2 Invoke nonnegativity criteria on those variables specified in IDNEG(*) vector with a flag of 1. The user must give the nonnegativity flags of all the variables in the system in the IDNEG(*) vector in the CUNIT data file.

IDISCN

Model discontinuity option. It indicates whether or not there is discontinuity in the model at any time during the simulation. If there is a discontinuity in the model, integration proceeds up to this time then control is returned to the Executive routine for further action. Values are as follows:

0 There is no discontinuity in the model at any time.

1 Discontinuity in the model occurs at time TDISCN. In this case the user is prompted to enter the time, TDISCN.

LCOMP

Components option. Indicates the nature of chemical species in the system to be solved, with values as follows:

0 Monocomponent system. In this case there is no VLE calculation during the simulation. No components data is required.

1 Multicomponent system. All units have the same number of components. The user will provide all the components data necessary for the calculation of K-values and/or enthalpies using the chosen method.

2 Multicomponent system. There are different number of components in each unit. The user will provide all the components data necessary for the calculation of K-values and/or enthalpies using the chosen method.

LPRNT

Communication time option. It indicates the time when output should be displayed. Values are as follows:

0 Output at the end of the simulation only.

1 Output at user-specified communication interval. The user is prompted for the communication interval. For dynamic simulation the integration time interval (HOUT) is required and for steady state simulation the number of iterations before display , NOUT.

2 Output at every step of integration or at every iteration.

LPLOT

Plotting option. Indicates whether plotting of variables is required or not, with values as follows:

0 No plotting is required.

1 Plotting is required using plotting routine no. 1. Information about the variables to plot must be supplied by the user in the general input data file.

2 Plotting is required using plotting routine no. 2. Information about the variables to plot must be supplied by the user in the general input data file.

LEVENT

Event processing option. This indicates whether or not events of any kind will occur during the simulation. Values are as follows:

0 No events will occur.

1 A scheduled state or time events will occur during the simulation. Information about the events must be supplied by the user in the general input data file.

Note that only events whose sequence of occurance are known apriori are accommodated.

MDEFLT

Integration step lengths option. Indicates whether the user will specify or the system should find appropriate initial step length and stepsize bounds for the numerical integration. Its values are as follows:

0 System should find appropriate step lengths and bounds.
1 User specifies the step lengths and bounds.

JFBS

Jacobian matrix structure option. This option indicates the structure of the Jacobian matrix of the equations to be solved and hence the type of algorithm to be used in the solution of the systems of linear equations. It has values as follows:

0 Full matrix structure. That is NxN matrix. Use dense matrix algorithms.
1 Banded matrix structure. In this case the lower, ML and upper, MU bandwidths of the matrix will be needed.
2 Sparse matrix structure. In this case a sparse matrix package is used to solve the linear equations, which requires the number of nonzeros, NZ, their equation numbers, IEQN(*), and their variable numbers, JVAR(*).

B7.3 USER AND PROJECT INFORMATION

TITLE

The title of the run. A character value of not more than 60 characters inclosed in quotes.

CUSER

Name of the user. A character value of not more than 12 characters inclosed in quotes.

CPROJ

Name of the project or problem to be solved. A character value of maximum length, 20 characters inclosed in quotes.

CDATE

Date of execution of project. A character value of maximum length, 12 characters enclosed in quotes.

B7.4 DATA FILE NAMES

CWORK

Variable name of the work file used by the DASP to store all the variable and their derivative values or the iterates of the variables at every step or iteration. The default value is "WKFILE".

CRESLT

Variable name of output file given by the user. It will contains all information about the simulation depending on the print level option requested. A character value of maximum length, 8 characters.

CINDAT

Variable name of the general input data file containing all the necesary data (see Appendix B2 for details). A character value of maximum length, 8 characters.

CTOPOL

Variable name of process topology data file containing all information about the topology of the problem (see Appendix B2 for details). A character value of maximum length, 8 characters.

COMDT

Variable name of the components data file containing all the data about the number, identity and all physico-chemical constants of the chemical species in the system (see Appendix B2 for details). A character value of maximum length, 8 characters.

CUNIT

Variable name of the units data file containing all information about the parameters and variables of all the units as well as the error tolerances. A character value of maximum length, 8 characters.

B7.5 OUTPUT INFORMATION

INDEX

Print level index. This indicates the level of output required according to the following values:

0 Output the user and problem descriptors, all equipment parameters and a specified number of variable values at the communication times.

1 As in INDEX=0 above plus the values of all input data.

2 Output all equipment parameter and all variable values plus all input data at communication times.

3 Output everything possible (as in INDEX=2 above plus Jacobian matrix values at every Jacobian evaluation and the values of the residuals of the equations, the variables and their derivatives at every function evaluation).

4 For the USRSUB Option only. The user will format his output within JS=3 and JS=5 section of the USRSUB routine.

LTERM

Terminal output control variable. It indicates whether or not the user want output to be displayed on the screen as well as in files. Values are as follows:

0 Interactive mode,

1 Output to be displayed on the terminal as well as in files.

NVPRNT

The total number of the specified variable values to print. This is limited to 10.

LUPRNT(*)

The unit labels of the specified variables to print. An integer array of length, NVPRNT.

LSPRNT(*)

For staged or sectionalized modules, the stage or section numbers of the specified variables. An integer array of length, NVPRNT.

LVPRNT(*)

The code numbers of the specified variables to print. An integer array of length, NVPRNT.

CPRNT(*)

The user-given names of the specified variables to print. A character array of length, NVPRNT (max. 12 characters per name) enclosed in quotes.

CUPRNT(*)

The units of the specified variables to print given by the user. A character array of length, NVPRNT (max. 12 characters per name) enclosed in quotes.

B7.6 PLOTTING INFORMATION

NVPLOT

The total number of variables to plot, limited to 5.

LUPLOT(*)

The units labels of the specified variables to plot. An integer array of length, NVPLOT.

LSPLOT(*)

For staged or sectionalized modules, the stage or section numbers of the specified variables to plot. An integer array of length, NVPLOT.

LVPLOT(*)

The code numbers of the specified variables to plot. An integer array of length, NVPLOT.

CPLOT(*)

The user-given names of the specified variables. A character array of length, NVPLOT (max. 12 characters per name) enclosed in quotes.

B7.7 NUMERICAL INTEGRATION INFORMATION

T0

The starting or initial time of simulation.

TFIN

The final or end time of simulation.

MFDYN

The method flag of the integrator to use for the numerical integration. The following values are applicable:

1 Use SDASSL integrator.

2 Use a user-written integrator via DYNSUB interface routine.

KDAE

Equation type flag. It indicates whether or not a distinction should be made between differential and algebraic variables. It has the following values:

0 There are only differential variables or no distinction should be made between differential and algebraic variables.

1 Differential and algebraic variables are distinguished during error test.

EDYN(*)

Contains flags of the nature of the variables in the system to be solved. These flags are as follws:

0.0 Variable is algebraic.

1.0 Variable is differential.

HMIN

The minimum stepsize to use. This sets the lower bounds of the step length.

H0

The initial step length to start the integration.

HMAX

The maximum step length. This sets the upper bound of the step length.

KORDX

The maximum order of the integration method specified. For the integrator DASSL, this is 5. Note a lower order can be specified if the user want to reduce the work space requirements of the integrator.

B7.8 ALGEBRAIC EQUATION SOLVER INFORMATION

KMAX

The maximum number of iterations to take towards the solution of algebraic equations.

MFALG

The method flag of the nonlinear equation solver to use for solving the algebraic equations. It has values as follows:

1 The Newton-Raphson equation solver routine.

2 Use Broyden's quasi-Newton's method.

3 User-written equation solver via ALGSUB interface routine.

4 Use a conbination of Newton-Levenberg-Marquart and Continuation method

routine.

LJAC

Availability of analytical Jacobian matrix indicator. This is a logical variable which has values as follows:

.TRUE. The analytical Jacobian matrix of the problem is provided.

.FALSE. No analytical Jacobian matrix is provided, use numerical Jacobian matrix option.

Note that all modules in DASP do not provide the analytical Jacobian.

KJAC

Initial Jacobian matrix type indicator for steady state simulation. It shows the type of Jacobian matrix to use at the starting period, with values as follows:

0 Use finite difference approximation to start the computation if LJAC=.FALSE.

1 Use identity matrix as initial Jacobian to start calculation and finite difference approximation for subsequent ones if LJAC = .FALSE.

ML

Lower bandwidth of the banded Jacobian matrix structure.

MU

Upper bandwidth of the banded Jacobian matrix structure.

NZ

The total number of nonzeros in the sparse Jacobian matrix.

IEQN(*)

The equation numbers of the nonzeros in the sparse Jacobian matrix. An integer array of length, NZ. This should be given in the order of increasing equation numbers.

JVAR(*)

The variable numbers of the nonzeros of the sparse Jacobian
matrix. An integer array of length, NZ. This is given equation by equation.

IDNEG(*)

Contains the nonnegativity flags of varaibles if NONNEG=2. The flags of the variables should be provided as follows:

= 0 if a variable can have a negative value

= 1 if a variable must remain nonnegative.

B7.9 EVENT PROCESSING INFORMATION

NEVENT

The total number of scheduled events ,both state and time events, which will occur during the simulation (max. 10).

KECODE(*)

The event codes, with values as follows:

1 Time event.

2 State event.

Integer array of length, NEVENT.

EVTIME(*)

The times when the events will occur. Note that for the case of state events, a value of zero should be given as the system will calculate the appropriate state event times. Real array of length, NEVENT.

KEUNIT(*)

The labels of the units of the variables whose threshold values will be used to determine state event times. A value of zero should be given for time events. Integer array of length, NEVENT.

KSUNIT(*)

For the case of staged or sectionalized modules, the stage or section numbers of the variables whose threshold values will be used to determine state event times. A value of zero should be given for time events. Integer array of length, NEVENT.

KVARNO(*)

The code numbers of the variables whose threshold values will be used to determine the state event times. A value of zero should be given for time events. Integer array of length, NEVENT.

TRSVAL(*)

The threshold values of the variables to be used to determine state event times. A value of zero should be given for time events. Real array of length, NEVENT.

TRSTOL(*)

The threshold tolerances for the variables whose threshold values will be used to

determine the state event times or to compare times for time events. A value of zero should be given for time events. Real array of length, NEVENT.

IDCROS(*)

The direction of threshold crossing. This specifies the direction in which the values of the specified variables, whose threshold values will be used to determine the state event times, are changing. If, in the region of the threshold value, the variable value is increasing then positive direction, otherwise negative direction. An integer array of length, NEVENT, with values as follows:

+1 positive threshold crossing.

-1 negative threshold crossing.

A value of zero should be given for time event cases.

CENAME(*)

User-supplied names of the event types. Character vector of maximum length 12 characters.

B7.10 VARIABLE AND PARAMETER INFORMATION

B7.10.1 GLOBAL VARIABLES

NVAR

Total number of variables in the system.

NEQN

Total number of equations in the system.

XVAR(*)

Array containing the values of all variables in the system. Real array of length, NVAR.

DXVAR(*)

Array containing the values of the derivatives of all variables in the system. Real array of length, NVAR.

FUNC(*)

Array containing the values of the residuals of all the equations in the system. Real array of length, NEQN.

B7.10.2 MODULE VARIABLES & PARAMETERS

MVAR

The total number of variables in a module.

MPMI

The total number of integer type parameters in a module or unit.

MPMR

The total number of real type parameters in a module or unit.

MFUNC

The total number of equations in a module.

X(*)

Array containing the values of the variables in a module. Real array of length, MVAR.

DX(*)

Array containing the values of the derivatives of the variables in a module. Real array of length, MVAR.

F(*)

Array containing the values of the residuals of the equatios in a module. Real array of length, MFUNC.

ICX(*)

Array containing the code numbers of the variables in a module. Integer array of length, MVAR.

IEP(*)

Array containing the values of the integer type parameters in a module or unit. Integer array of length, MPMI

REP(*)

Array containing the values of the real type parameters in a module or unit. Real array of length, MPMR.

ICEP(*)

Array containing the values of the code numbers of the real type parameters in a module. Integer array of length, MPMR.

B7.11 ERROR TOLERANCE INFORMATION

TOL

Scalar relative error tolerance. Real value.

RTOL(*)

Vector relative error tolerances. Real values of length, MVAR (for each unit) or NVAR (for the whole system to be solved).

ATOL(*)

Vector absolute error tolerances. Real values of length, MVAR (for each unit) or NVAR (for the whole system to be solved).

B7.12 COMPONENTS INFORMATION

NCX

The total number of components in the problem to be solved (max. 20 components).

MFK

VLE package method flag. This flag indicates which VLE package to use for the calculation of the K-values. It has values as follows:

1 Use the UNIQUAC model.
2 Use the UNIFAC model.
3 Use POLYNOMIAL curve fits.
4 Use user-supplied VLE package via KVAL04 interface routine.

MFH

Enthalpy calculation package method flag. This flag indicates which ENTHALPY package to use for the calculation of the components and total enthalpies. It has values as follows:

1 Use the UNIQUAC model.
2 Use the UNIFAC model.
3 Use POLYNOMIAL curve fits.
4 Use user-supplied ENTHALPY package via ENTH04 interface routine.

IDEAL

System ideality flag. It indicates whether or not the system is an ideal one. It has values as follows:

0 An ideal system.
1 A nonideal system. Note that for an ideal system, the enthalpy of mixing is

neglected and the activity coefficients will be assumed unity.

IDX(*)

The identity of all the components in the system, given serialy starting from 1. Integer array of length, NCX.

DMWT(*)

The molecular weights of the components in the system. Real array of length, NCX.

NC

The number of components in a module.

ID(*)

The identity of the components in a module. Integer array of length, NC.

TC(*)

The critical temperatures of the components in degree Kelvin (K). Real array of length, NCX.

PC(*)

The critical pressures of the components in bars. Real array of length, NCX.

DM(*)

The dipole moments of the components in debyes (D). Real array of length, NCX.

RD(*)

The radius of gyration of the components in Angstroms (A). Real array of length, NCX.

ZR(*)

The racket equation parameters of the components. Real array of length, NCX.

HVAP(*)

The heats of vaporization of the components in J/mol. Real array of length, NCX.

RU(*)

The structural volume parameter for the UNIQUAC equation, $\{r_i\}$, of the components. Real array of length, NCX.

QU(*)

The structural area parameter for the UNIQUAC equation, $\{q_i\}$, of the components. Real array of length, NCX.

QP(*)

The modified structural area parameter for the UNIQUAC equation, $\{q'_i\}$, of the components. Real array of length, NCX.

ETA(*)

The association and solvation parameters of the components. Real array of length, NCX*(NCX+1)/2.

U(I,J)

Array of UNIQUAC binary interaction parameters (set to 0.0 if not available, U is zero for condensable components, and 10^{20} for noncondensable conponents).

AVP(*)

The Antoine constants, A of the components in the equation of the form ln(P)=A-B/(T+C) . Real array of length, NCX.

BVP(*)

The Antoine constants, B of the components in the equation of the form ln(P)=A-B/(T+C) . Real array of length, NCX.

CVP(*)

The Antoine constants, C of the components in the equation of the form ln(P)=A-B/(T+C) . Real array of length, NCX.

NG

The total number of unique groups in the system. It is used for the UNIFAC model. Maximum number of groups is 10.

IDGP(*)

The identity of the unique groups in the system. It is used for the UNIFAC model. Integer array of length, NG.

NG1

The total number of unique groups in component, I.

ITAB(I,*)

The group identity of the unique groups in component, I. Integer array of length, NG1.

NY(I,*)

The number of repititions of the unique groups in component, I. Integer array of length, NG1.

C1(*), C2(*), C3(*), C4(*), C5(*)

The constants for zero pressure reference fuggacity equation for all components. The equation is of the form ln(F0) = C1 + C2/T + C3*T + C4*ln(T) + C5*T**2 (T in K, F0 in bars).

D1(*), D2(*), D3(*), D4(*)

The constants for ideal gas heat capacity equation for all components. The equation is of the form CP0 = D1 + D2/T + D3*T + D4*ln(T) (T in K, CP0 in J/Deg-Mol).

CV1(*), CV2(*), CV3(*), CV4(*), CV5(*)

The constants for gas heat capacity equation for all components. The equation is of the form CP = CV1 + CV2*T + CV3*T**2 + CV4*T**3 + CV5*T**4 (T in K, CP in J/Deg-Mol)

CL1(*), CL2(*), CL3(*), CL4(*), CL5(*)

The constants for liquid heat capacity equation for all components. The equation is of the form CP = CL1 + CL2*T + CL3*T**2 + CL4*T**3 + CL5*T**4 (T in K, CP in J/Deg-Mol).

APPENDIX B8
VARIABLE AND PARAMETER TYPES

B8.1 INTRODUCTION

This section presents the variable and parameter types defined in DASP. At present 100 variable and parameter types have been defined and each is given a unique positive code number and a name. In order to handle an equilibrium stage as a unit module, the vapor and liquid variable types are distinguished. Also the units which can be used to input and output the values of these variables and parameters are given. Two types of units have been defined, namely SI and British Units. The user can only use one type of units for all the variables and parameters as there is no conversion between units. Note that the distinction between variables and parameters is arbitrary, as this may depend on the modelling assumptions made and what the model is intended to be used for. Also, the code numbers of the integer type parameters are also given. This was intended to be used in conjunction with an input language.

B8.2 VARIABLE TYPES

Variable Type	Variable Name	Code Number	SI Unit	British Unit
Vapor mole fractions	YCOMP	1-20	$\frac{\text{kgmole}}{\text{kgmole}}$	$\frac{\text{lbmole}}{\text{lbmole}}$
Liquid mole fractions	XCOMP	21-40	$\frac{\text{kgmole}}{\text{kgmole}}$	$\frac{\text{lbmole}}{\text{lbmole}}$
Vapor holdup	VHOLD	41	kgmol	lbmole
Liquid holdup	LHOLD	42	kgmole	lbmole
Total vapor flow rate	VRATE	43	kgmole/hr	lbmole/hr

Variable Type	Variable Name	Code Number	SI Unit	British Unit
Total liquid flow rate	LRATE	44	kgmole/hr	lbmole/hr
Vapor volume	VVOL	45	m^3	ft^3
Liquid volume	LVOL	46	m^3	ft^3
Vapor density	VDENS	47	$kgmole/m^3$	$lbmole/ft^3$
Liquid density	LDENS	48	$kgmole/m^3$	$lbmole/ft^3$
Liquid level	LHGHT	49	m	ft
Pressure	PRES	50	Pa	atm
Temperature	TEMP	51	K	°F
Vapor enthalpy	VENTH	52	kJ/kgmole	Btu/lbmole
Liquid enthalpy	LENTH	53	kJ/kgmole	Btu/lbmole
Percentage	PCENT	54	%	%
Pneumatic control signal	PSIGN	55	psig	psig
Electronic signal	ESIGN	56	mA	mA
Heat load	HTLOAD	57	kJ/hr	Btu/hr
Unspecified variable	VARIABLE	58-60	-	-

B8.3 PARAMETER TYPES

Parameter Type	Parameter Name	Code Number	SI Units	British Units
Zero of input	ZI	61	Same as the input variable	Same as the input variable
Range of input	RI	62	Same as the input variable	Same as the input variable
Zero of output	ZO	63	Same as the output variable	Same as the output variable
Range of output	RO	64	Same as the output variable	Same as the output variable
Time constant	TAU	65	hrs	hrs
Zero offset of input	X0	66	Same as input variable	Same as input variable
Bias or manual reset	Y0	67	Same as output variable	Same as output variable
Controller action +1 for direct -1 for reverse	AXN	68	-	-
System or process gain	GAIN	69	$\frac{\text{Unit of output}}{\text{Unit of input}}$	$\frac{\text{unit of output}}{\text{unit of input}}$
Set point of controller	SP	70	Same as input variable	Same as input variable

Parameter Type	Parameter Name	Code Number	SI Units	British Units
Integral time	TI	71	hrs	hrs
Derivative time	TD	72	hrs	hrs
Damping ratio	DAMP	73	-	-
Value constant	CV	74	kgmole/hr Pa^2	lbmole/hratm^2
Fractional valve opening	AV	75	-	-
Equal % value trim constant	ALFV	76	-	-
Individual/ composite heat transfer coefficient	HIO	77	kJ/hr m^2 K	Btu/hr ft^2 °F
Overall heat transfer coefficient	UHTC	78	kJ/hr m^2 K	Btu/hr ft^2 °F
Diameter	DIAM	79	m	ft
Area	AREA	80	m^2	ft^2
Efficiency	EFF	81	-	-
Ratio (eg. reflux or split fraction)	RATIO	82	-	-
Mass of metal, material etc	MASS	83	kg	lb

Parameter Type	Parameter Name	Code Number	SI Units	British Units
Molecular weight	MOLWT	84	-	-
Heat capacity	CP	85	kJ/kgmole-K	Btu/lbmole-°F
Latent heat of vaporization	HVAP	86	kJ/kgmole	Btu/lbmole
Heat of reaction	HTR	87		
Controller manual setting	CMAN	88	Same as output variable	Same as output variable
Number of revolutions per minute	RPM	89	1/min	1/min
Unspecified parameter	PARAMETER	90-100	-	-

B8.4 INTEGER PARAMETERS

Parameter Name	Code Number	Description
MOPTN	1	The model option variable, which gives the various options based on the modelling assumptions, eg. isothermal or adiabatic option.
MPARAM	2	The parameter option variable, which gives the various options available for a specified parameter, eg. set point is constant or variable.

Parameter Name	Code Number	Description
ICODEI	3	Code number of input variable.
ICODEO	4	Code number of output variable.
IDERIV	5	Derivative availability option, which has values as follows: = 1 derivatives of variable supplied, = 0 derivatives not available.
NIS	8	No of input streams.
NOS	9	No of output strams.
NR	10	No of reaction paths
JOPTN	11	Controller mode option

APPENDIX 9
ERROR MESSAGES

1. The amount of integer type workspace available for this problem is insufficient. The array ICORE should be increased as indicated in the accompanying message.

2. The amount of real type workspace available for this problem is insufficient. The array CORE should be increased as indicated in the accompanying message.

3. The number of vaiables/equations in the system to be solved should be greater than zero.

4. The error tolerance(s) specified are too stringent. As a guide , they should be greater than the machine round-off number as given in the message above.

5. The local error test during the numerical calculation could not be satisfied, probably because a zero component was specified in both the relative (RTOL) and absolute (ATOL) error tolerances.

6. I/O error. An end of file occured during an input/output operation. This normally happens when data is being read from a sequential file without first rewinding the file.

7. I/O error. The system was unable to access a file for input/output. Maybe the file does no exist or has not been properly connected or the access mode used was wrong.

8. File connection error. The file whose logical unit number is given above could not be connected for input/output. Either the file does not exist when it should or vice visa or the mode of access is wrong.

9. The initial derivatives of the state variables could not be computed by the code. This could happen when the initial approximations to the derivatives are not good enough or the derivatives consistent with the initial values given do not exist.

10. The specified number(KMAX) of iterations has been taken without convergence. You may have another go for another KMAX number of iterations.

11. The code has taken about 500 steps in the numerical integration towards the communication interval (TOUT) without reaching there. You may have another go for

another 500 steps.

12. Incorrect input value(s) were detected which were outside the range of values for these variable(s).

13. Incorrect input. The permissible number of incorrect input trials has been exceeded. Bye.

14. The matrix of partial derivatives is singular. Maybe some of the equations are redundant or there is no solution to this problem or the solution is not unique.

15. The function values could not be computed by the code. Maybe some of the iterates/state variable values will cause a division by zero or overflow or underflow, etc.

16. The Jacobian values could not be computed by the code. Maybe some of the iterates/state variable values will cause a division by zero or overflow or underflow etc.

17. Convergence problem. Repeated error test failures occured during the last test in the numerical integration. This may be due to singularity problem of the Jacobian matrix.

18. There was a multiple convergence test failures preceeded by multiple error test failures on the last attempted step. It is possible that the problem is ill-posed or cannot be solved by this code or there is singularity/discontinuity in the problem.

19. The function/Jacobian values were crudely approximated for the last step and as a result there was repeated error test failures for the last attempted step.

20. An error condition was reported but nothing was done to remedy it before the code was called again. You caanot continue the execution in this situation.

21. The value of a variable/parameter could not be located in a storage location. This may happen when there wrong input data was given especially during topology or variable code numbers input.

22. A dummy subroutine was called/referenced by the code. This option is not yet available.

23. The code was unable to set up the sparse matrix structure. Maybe some of the module routines do not include this option. The dense matrix structure has been assumed but this may required more work space than is available.

24. There is error in the sparse Jacobian matrix structure which became apparent when equation/variable numbers were referenced. You may restart the execution after correcting the mistake or use the dense matrix option.

25. The code has modified a user input value due to wrong input. Further computations make use of the new value.

26. No further message included for this type of error.

APPENDIX C

AN ALTERNATIVE IMPLEMENTATION OF DASP

LIST OF CONTENTS FOR APPENDIX C

		PAGE NO
C1	Introduction	292
C2	The Model Routine	292
C2.1	The Structure of a Module	292
C2.2	Initialization Section	294
C2.3	Function Evaluation Section	296
C2.4	Jacobian Evaluation Section	296
C3	Equation Generation	296
C4	Transfer of Variables between Modules	298
C5	The USRSUB Option	299
C6	Event Processing	300

C1 INTRODUCTION

The implementation of DASP presented so far contains some arbitrarily imposed restrictions and limitations, which tend to limit the main advantages of an equation-oriented simulator. For example, the division of all real variables into parameters and variables is arbitrary. As a result of experience in developing and testing the ideas in DASP a systematic and coherent implementation strategy can be proposed. In this chapter, such an implementation strategy is described which will take full advantage of the equation solving approach. The other main advantages are that it is simpler to understand, and implement, and would reduce the Fortran code by a substantial amount. These advantages are emphasised at the relevant sections.

C2 THE MODEL ROUTINE

The model routine described in Chapter 6 has a structure with functional divisions of initial, function evaluation, Jacobian evaluation, output, event, terminal and nonzeros of Jacobian matrix setup sections. The difference is in the actual implementation of these sections. The division of the real variables into parameters and variables is arbitrary and tend to negate the major advantage of equation-oriented simulation, where any variable or parameter can be regarded as the unknown in the system. If we remove this unncessary restriction, we have a model routine which, as described below, is much more flexible. The various sub-sections will be described with reference to a controller module, which models a P, PI and PID control.

C2.1 THE STRUCTURE OF A MODULE

The structure of the model routine is the same as described in Chapter 6. However, the number and functions of the argument lists and common block variables are different. The subroutine has the following format:

```
SUBROUTINE MODULE (IC, JS, T, X, DX, ICX, ICDE, IEP,
                   F, CJ, IRES, RP, IP, CR, ICR)
```

where

IC is the code number of the module

JS is the section control variable (same as described in Chapter 6)

T is the time

X (*) is the vector which contains all the real variables in the module (both known and unknown)

DX (*) is a vector of the derivatives of the unknown variables

ICX (*) is a vector containing the integer flags, which shows which variables

are known (= 0) and which are unknown (= 1)

ICDE(*) is an integer vector, which contains the code numbers of all the real variables

IEP (*) is a vector which contains the integer parameters in the module

F(*) is a vector which stores the residual of the equations calculated during function evaluation or the sub-Jacobian matrix for this module during Jacobian evaluation.

CJ is a real value used in Jacobian matrix calculation.

IRES is a performance flag, set to a negative value of the error code number if any errors occur in the module.

RP is a real value which may be used in the module. Not modified by DASP.

IP is an integer value which may be used in the module. Not modified by DASP.

CR(*) is a real vector, which is used as work space in DASP. It must not be used in the module.

ICR(*) is an integer vector used as work space by DASP. It must not be used in the module.

In this subroutine, any of the real variables in the module can in principle be an unknown variable just by changing the flag of the variable in ICX (*) vector. The common block, MODWK1, is modified to reflect the new structure:

```
COMMON /MODWK1/ IU, LDNIN, LDNMES, MVAR, MFUNC, MPMI,
                IUOPT, IUNC, IOPTN, NC, ID(20), IWK(20), LIN, LOUT
```

where

IU is the index of this module in the vector of unit numbers, IUNOS(*), of all the units in the blockdiagram of the flowsheet.

LDNIN is the input channel number for the input data file

LDNMES is the channel number of the file which contains messages such as code numbers of all the variables in a module

MVAR is the total number of real variables in the module

MFUNC is the total number of equations in the module

MPMI is the total number of integer variables in the module

IUOPT is the flag which indicates the units used by the user to input all the variable values (eg. 1 for SI and 2 for British Unit)

IUNC is an integer value, which on input contains the unit number of this module. However, when the retrieve routine, RTRV is called during function or Jacobian evaluation, it returns the unit number of the unit from which the input variables to this module are retrieved. If the value of IUNC

is negative, it signifies that that unit has been disconnected from the block diagram of the flowsheet.

IOPTN is an integer value, which indicates the type of simulation being carried out (eg. 1 for steady state, 2 for dynamic simulation).

NC the number of chemical species in the module

ID(*) is a vector of the identity of the species in the module

IWK(*) is an integer workspace

LIN is the channel number for input from the keyboard

LOUT is the channel number for output to the terminal

A common block of real workspace which can be used in the module is also needed. A suitable one is:

```
COMMON /MODWK2/ FX(100), DFX(100), RPM(100)
```

where

FX(*) is a real vector for retrieving the values of input variables

DFX(*) is a real vector for retrieving the values of the derivatives of input variables

RPM(*) is a real vector used as workspace

C2.2 INITIALIZATION SECTION

This section (JS = 0) initializes the module for simulation. The first data item to set up is the integer parameters which determine the model and parameter options in the module. These data items are described for each module in the DASP Module Library. The integer parameters for the controller module are:

1 MOPTN:
= 1 for P-control
= 2 for PI-control
= 3 for PID-control

2 MPARAM:
= 0 for constant setpoint
= 1 for variable setpoint (ie. input from another unit)

3 ICODEI code number of input signal to the controller

4 ICODEO code number of controller output

5 JOPTN Mode of controller

= 0 for automatic mode

= 1 for manual mode

This data will be read from the input file via this implementation

```
      IRES = 0
      IF (JS . EQ . 0) THEN
C  ----  Initialization section
         IUNC = 1
         MPMI = 5
         IWK(1) = 1
         IWK(2) = 2
         IWK(3) = 3
         IWK(4) = 4
         IWK(5) = 11
         CALL UNIPMI (LDNIN, IEP, IWK, MPMI, IRES)
```

C -- Check for input error and correctness of input values and write the appropriate
C -- message.

The real variables in the module include all the parameters, bounds on variables, the known and unknown variables. For the controller module these are:

1	ZI	2	RI
3	ZO	4	RO
5	Y0	6	AXN
7	GAIN	8	CMAN
9	TI	10	TD
11	SP	12	PSIGN or ESIGN

The description of these variables is given in Appendix B8. The default unknown variable here is PSIGN. However, in other situations the gain, GAIN can be selected instead of PSIGN. The code numbers of these variables can be read from a module's data file, with channel number, LDNMES, which contains the information about the modules in some specified order or format. This could be read via a Fortran subroutine call of the form

CALL UNIMES (LDNMES, ICDE, MVAR, IRES, KEY)

where KEY signifies the type of information to read. All the real variables in the module are read via a call to subroutine UNIPMR as in

CALL UNIPMR (LDNIN, X, ICDE, MVAR, IRES)

The user is expected to choose which of the above variables are known and which are unknown. A known variable is given a flag of 0 while an unknown variable is given a

flag of 1 in the array, ICX (*) via a call to INPUTI routine as in

CALL INPUTI (LDNIN, ICX, MVAR, IRES)

Naturally, the correctness of the input values must be verified and for any errors detected an appropriate message is output for the user. For dynamic simulation, there is no need to ask the user to give the derivatives of those variables which have derivative terms since they will be determined as described in Chapter 7. The main advantage of the structuring of the modules in this way is that all modules have a common structure, and does not depend on type. This means that the modules are regarded as "black boxes" by the Executive subprogram. Also having mastered the structure of one module, other modules can be written in the same way.

Also, it is not necessary to choose the variables from among the real variables of the particular module where the equations are coded. Input variables can be chosen as the unknown in the equation. This is most useful in steady state design.

C2.3 FUNCTION EVALUATION SECTION

This section is called if JS=1. It retrieves the values of the input variables from the connected units and then calculates the residuals of the equations. The procedure is almost the same as described in Chapter 6 and Appendix 5.

C2.4 JACOBIAN EVALUATION SECTION

It is difficult to set up analytical Jacobian matrix of the problem, especially now that any variable in the module can be regarded as an unknown. Hence the Jacobian may have to be calculated numerically.

All the other sections will be set up as described in Chapter 6 and Appendix 5.

C3 EQUATION GENERATION

The equations are generated as the residuals of each equation, calculated module by module, given the values of the input variables (retrieved from connected units) and those of the variables and derivatives of the module.

During the initialization region, all the values of variables, their code numbers, the flags of the variables (known or unknown) are read from the input data file. These values are stored in various sections of the vectors CORE (*) and ICORE (*) as shown below.

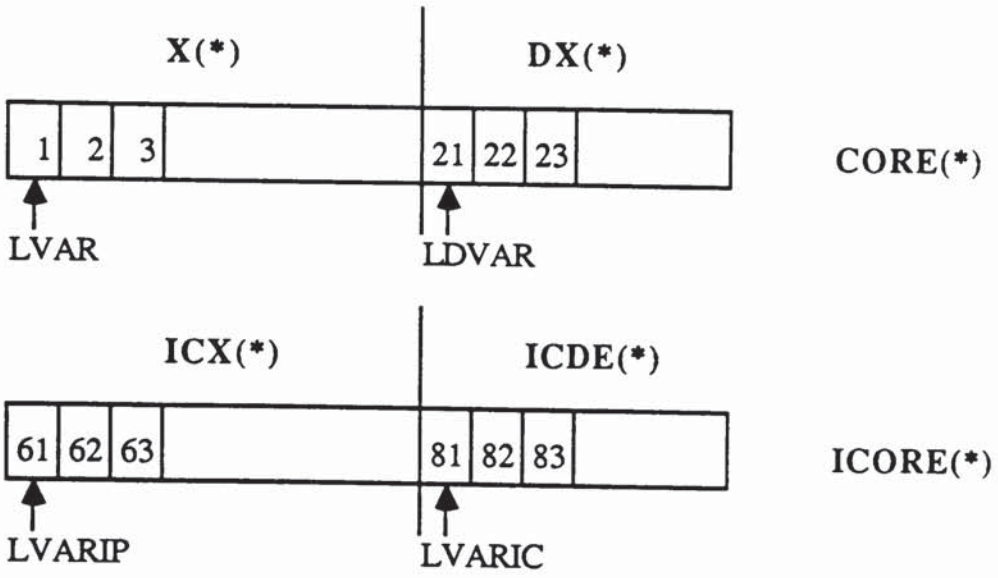


Figure C1 - Arrangement of Variables in Work Spaces CORE(*) and ICORE(*)

In figure C1, the variables values, X(*) for a module are stored in CORE(*) vector starting from position 1 which is stored as the pointer, LVAR in the Figure. These starting positions are stored in a storage location, LOCVAR(*) vector in position equal to the index of the module.

Not only is it necessary to call the model routine with the correct variables associated with that model routine, but also to retrieve the correct input variables values to that module from other connected modules. This is illustrated in Figure C2.

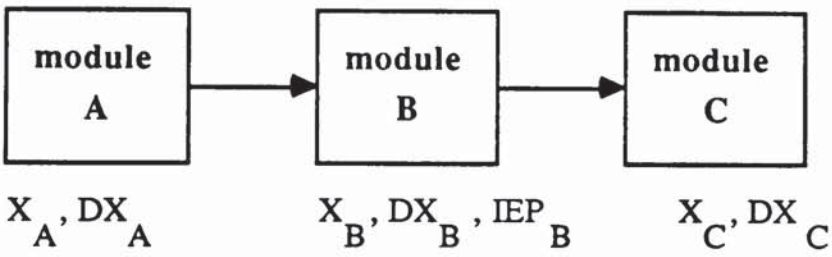


Figure C2 - Association of Variables between Modules

In each module, the variables X, DX and IEP are termed variables of the module. During function evaluation, these variables are passed to the module via the argument lists of the module subroutine. In Figure C2 Module B is being called to calculate the residuals of the equations modelling the module. To pass the values of variables of the module via the argument list of the subroutine, all that is needed is the index of the module which is the position of its unit number in IUNOS (*) vector. The starting position of the storage location in CORE(*) vector is got from LOCVAR(*) vector. This process is used to store all other variables and parameters in the CORE(*) and ICORE(*)

vectors. The number of variables in a module are stored in NVARU(*) vector using the module index. All the storage locations are set up during the initial region by a special routine.

The total number of unknowns in the system and their derivatives are extracted from X(*) vector using the flags in ICX(*). These unknown variables are stored in new arrays XVAR(*) and DXVAR(*) serially, module by module and their positions in X(*) vector stored in IPXVAR(*) vector. The integrator and the nonlinear equation solvers use of XVAR(*) and DXVAR(*) only.

Note that before the modules are called for both function and Jacobian evaluations, it is necessary to update the values of the unknown variables and their derivatives in X(*) and DX(*) vectors using the recent values forwarded by the integrator in XVAR(*) and DXVAR(*) respectively. A suitable subroutine to do this could be:

```
      SUBROUTINE  RESET (X, DX, XVAR, DXVAR, IPXVAR, NVAR)
      DOUBLE PRECISION X (*), DX (*), XVAR (*), DXVAR (*)
      INTEGER IPXVAR (*), NVAR
      DO 100 I=1, NVAR
          X (IPXVAR(I)) = XVAR(I)
          DX (IPXVAR(I)) = DXVAR(I)
100    CONTINUE
      RETURN
      END
```

C4 TRANSFER OF VARIABLES BETWEEN MODULES

During function and Jacobian evaluations the pointers to the first positions of the variables and derivatives of the module are got from LOCVAR(*) and LOCDX(*) vectors respectively. Let these pointers be $LVAR_B$ and $LDVAR_B$. Also the starting position $LPMI_B$ and $LFUNC_B$ of the integer variables and the equations of the module respectively are got from LOCPMI(*) and LOCEQN(*) respectively. Using these pointers, the module subroutine is called with

$$X(LVAR_B), DX(LDVAR_B), IEP(LPMI_B), F(LFUNC_B)$$

transferred to the module via the argument lists. In figure C2, module B is connected to module A and module C. This information is stored as follows:

$MP(IU_B, 1)$ = unit number of module A

$MP(IU_B, 2)$ = unit number of module C

where IU_B is the index of module B and MP(*,*) is a matrix, which stores the connectivity information

To retrieve the input variables and derivatives of the input module A for use in the equations of module B, we determine the index of module A, IU_A and this is used to retrieve X_A and DX_A which are forwarded to module B via FX(*) and DFX(*) respectively, in such a way that a variable with code number 51 is stored in position 51 of FX(*) and the derivative in position 51 of DFX(*). Thus together with X_B and DX_B, the residuals of the equations in module B are calculated and returned in F(*), which is a subset of the residual vector, FUNC(*).

C5 THE USRSUB OPTION

The USRSUB option (see Appendix B4) provides a means for the user to use his own model equations instead of using model routines in DASP. It is the equivalent of the fully equation-oriented approach as opposed to the modularly organized equation-oriented approach described with regard to the use of the model routines library. However at present, the user can only use one approach at a time and no attempt was made to unify them so that it is possible to assemble some of the equations from model routines and others via the use of the USRSUB routine. Although the user can add his own model routine if the particular routine is not available in DASP library, it must conform to the format described for a model routine (see Appendix B5). Thus the flexibility offered by the USRSUB routine cannot be utilized.

One of the advantages of the use of the USRSUB option is that subroutines of procedures of the sequential modular approach can be combined very easily with equations written for the equation-oriented simulation. A simple example is the calculation of the enthalpy of a mixture via the use of enthalpy calculation routines. Instead of writing the equations in the form

$$h = h\,(T, P, X)$$

where

h = enthalpy of the mixture

T = temperature

P = pressure

X = mole fractions

We can write

$$r = h(T, P, X) - h$$

where r is the residual of the equation. Thus h becomes one of the global variables instead of intermediate variable.

We can apply the same principle of section C2, where any of the variables and parameters can be regarded as an unknown to be calculated. Thus all these variables will be stored in X(*) and their derivatives in DX(*). An array of flags of the known and unknown will be stored in ICX(*). The subroutine will still maintain its structure as described in Appendix B4 but the argument list will be as follows:

```
SUBROUTINE USRSUB (LIN, LOUT, JS, T, X, DX, ICX, F, PD,
                   MVAR, MFUNC, CJ, IRES, RPAR, IPAR)
```

where

MVAR is the total number of real variables in the system

MFUNC is the total number of equations in the system.

To combine the equations generated by the use of the model routines with those from the USRSUB routine, we need to devise a new value for the source of model routine flag, KMODEL. Thus a value of 3 for KMODEL could mean generation of equation from both model routine and USRSUB. This could cause complications in transferring variables between the modules and the USRSUB, which has to be solved.

C6 EVENT PROCESSING

In Chapter 8, a description of how DASP handles events is presented. The main limitation was that only events whose sequence of occurrence is known apriori is accommodated and secondly it is assumed that no multiple events occur. For example, it may be required to run a simulation until either the concentration of a component has reached a threshold value or an amount of time has elapsed. This means that when one of these events happens, the other is not needed. To handle events of this type, the modifications needed in the description in Chapter 8 is to declare the vectors containing information about these events as two dimensional arrays. We can still maintain the requirement that events happen in a particular sequence but remove the restriction that only one event can happen at an event time. Thus multiple events may be specified to happen and the integrator checks the threshold crossing for each case. An algorithm

which may be used is as follows:

(1) After every prediction phase of integration, check if the threshold values have been crossed for any of the possible events in a set.

(2) If yes, proceed to (3). Otherwise exit to the integrator.

(3) Set the event flag to TRUE. Determine the event time as decribed in Chapter 8. Integrate from the previous time to the new event time and return control to the event code processing package.

REFERENCES

Alexander R (1977), "Diagonally Implicit Runge Kutta Methods for Stiff ODEs", SIAM J Num Anal 14, 1006-1022.

Anon (1967), "The Continuous System Simulation Language (CSSL)", Simulation, Vol 9, No 6, December, 1967.

Anon (1970), Union of Japanese Scientists and Engineers", JUSE-L-FIGS - Generalised interrelated Flow Simulation Program", User's Manual (2nd Edition), Computation Centre, Shibuya-Ku, Tokyo, Japan.

Anon (1974), ICL SLAM Manual, International Computers Ltd, UK, 1974.

Aylott MR, Ponton JW and Lott DH (1985), "Development of a Dynamic Flowsheeting Program", IChE Symposium Series No 92, pp 55-66, 1985.

Babcock PD (1982), "Application of an Equation-Oriented Flowsheet Simulator to Dynamic Simulation of Chemical Processes", Proceeding of the 1982 Summer Simulation Conference, pp 537-541, Simulation Councils Inc, 1982.

Barney JR (1975), "Dynamic Simulation of Large Stiff Systems in a Modular Simulation Framework", PhD Thesis, University of Western Ontario.

Barney JR, Ahluwalia RS and Johnson A I (1975), "DYNSYS 2.0 User Manual", University of Western Ontario, London, Canada.

Berger F and Perris FA (1979), "FLOWPACK II - A new Generation of System for Steady State Process Flowsheeting", Comput Chem Eng Vol 3, pp 309-317,1979.

Berna TJ, Locke MH and Westerberg AW (1980), "A new Approach to Optimisation of Chemical Processes", AIChEJ, Vol 26, No 1, pp 37-43, 1980.

Bobrow S, Ponton JW and Johnson AI (1971), "Simulation of the Transient Behaviour of Complex Chemical Plants using a Modular Approach", The Can Journal of Chem Eng Vol 49, 391-397, 1971.

Brannock NF, Vernevil VS & Wang YL (1979), "PROCESS™ Simulation Program - an Advanced Flowsheeting Tool for Chemical Engineering", CACE '79, EFCE, Montreux, April 8-11, 1979.

Brayton RK, Gustavson FG and Hachtel GD (1972), "A new Efficient Algorithm for Solving Differential-Algebraic Systems using Implicit Backward Differentiation Formulas", Proc IEEE Vol 60 No 1, pp 98-108, 1972.

Brennam RD and Liverbarger RN (1964), "A Survey of Digital Simulation", Simulation Vol 3, No 6, December 1964.

Briggs DE (1974), "DISCO - An Interactive Executive Program for Dynamic Simulation and Control of Chemical Processes", 78th National AIChE meeting, Salt Lake City.

Brown RL and Gear CW (1973), "Documentation for DFASUB - A Program for the Solution of Simultaneous Implicit Differential and Nonlinear Equations", Report

UIUCDCS-R-73-575, University of Illinois at Urbana - Champaign, 1973.
Bui TD (1981), "Solving Stiff Differential Equations in the Simulation of Physical Systems", Simulation, Vol 37, August 1981, pp 37-46.

Burden RL, Faires JD & Reynolds AC (1981), "Numerical Analysis", Second Edition, PWS Publishers, Massachusetts, USA, 1981.

Burka MK (1982), "Solution of Stiff ODEs by Decomposition and Orthogonal Collocation", AIChE Journal, Vol 28, No 1, pp 11-20, 1982.

Byrne GD and Hindmarsh AC (1975), "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations", ACM Trans Math Software Vol 1 No 1, pp 71-96, 1975.

Byrne CD, Hindmarsh AC, Jackson KR and Brown HG (1977), "A Comparison of two ODE Codes: GEAR and EPISODE", Comput Chem Eng Vol 1, pp 133-147, 1977

Cameron IT (1981), "Numerical Solution of Differential-Algebraic Systems in Process Dynamics", PhD Thesis, University of London, 1981.

Cameron IT (1983) "Large Scale Transient Analysis of Processes - The State-of-the-art Review", Latin American Journal of Chemical Engineering and Applied Chemistry", Vol 13, pp 215-228, 1983.

Campbell SL and Petzold L (1982), "Canonical Forms and Solvable Singular Systems of Differential Equations", SIAM J Alg Disc Meth, 4, pp517-521.

Caracotsios M and Stewart WE (1985), "Sensitivity Analysis of Initial Value Problems with Mixed ODEs and Algebraic Equations", Comput Chem Eng, Vol 9, No 4, pp359-365.

Carnahan B and Wilkes JO (1980), "Numerical Solution of Differential Equations - an Overview" in Foundations of Computer-aided Chemical Process Design". Henniker, New Hampshire, July 9, 1980.

Carvar MB (1978), "Efficient Integration Over Discontinuiti4es in ODE Simulations", Math and Comput in Simulation XX, pp190-196, 1978.

Carvar MB (1981), "Method of Lines Solution of Differential Equations - Fundamental Principles and Recent Extensions", in Mah RSH and Seider WD (eds), Foundations of Computer-aided Chemical Process Design, Vol I, p 369, Engineering Foundation, NY.

Chan YNI, Birnbaum I and Lapidus L (1978), "Solution of Stiff Differential Equations and the use of Embedding Technique (review)", Ind Eng Chem Fundam, Vol 17 No3, pp 133-148, 1978.

Chen HS and Stadtherr MA (1981), "A Modification of Powell's Dogleg Method for Solving Systems of Nonlinear Equations", Comput Chem Eng, 5, p143.

Chen H and Stadtherr MA (1985), "A Simultaneous Modular Approach to Process Flowsheeting and Optimization Part I and II", AIChE J Vol 31, No 11, pp 1843-1881, November, 1985.

Clark SM and Reklaitis GV (1984), "Investigation of Strategies for Executing Sequential Modular Simulations", Comput Chem Eng, Vol 8, No 3/4, pp 205-218, 1984

Crowe CM (1984), "On the Relationship Between Quasi-Newton and Dominant Eigenvalue Methods for the Numerical Solution of Nonlinear Equations", Comput Chem Eng Vol 8, No 1, pp 35-41, 1984.

Curtis AR (1978), "The FACSIMILE Numerical Integrator for Stiff Initial Value Problems, Harwell Report AERE-R-9352, 1978.

Davis ME (1984), "Numerical Methods and Modelling for Chemical Engineers", John Wiley and Sons Inc, USA, 1984.

Dongarra JJ, Bunch JR, Moler CB and Steward GW (1979), "LINPACK User's Guide", SIAM, Philadelphia.

Duff IS (1977), "MA28 - A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations", AERE Report R.8730, HMSO, London 1977.

Edsberg L (1981), "Some Numerical Problems in Mathematical Models for Chemical Kinetics", Studies in Physical and Theoretical Chemistry (1981), Vol 16 (Data Process Chem), pp 88-95.

Ellison D (1981), "Efficient Automatic Integration of ODEs with Discontinuities", Math and Comput in Simul XXIII, pp12-20, 1981.

Enright WH, Hull TE and Lindberg B (1975), "Comparing Numerical Methods for Stiff System of ODEs" BIT Vol 15, pp 10-48, 1975.

Evans LB (1980) "Advances in Process Flowsheeting Systems", in Mah R SH and Seider WD (Eds), Foundations of Computer-aided Chemical Process Design, Proceedings of an International Conference, Henniker, New Hampshire, July 1980.

Evans LB, Boston JF, Britt HI, Gallier K, Gupta K, Joseph B, Mahalec V, Ng E, Seider WD and Yagi H (1979), "ASPEN - An Advanced System for Process Engineering", Proceedings of the CACE '79 EFCE Symp, Montreux, Switzerland, April 8-11, 1979.

Fagley JC and Carnahan B (1982), "Comparison of Integration and Equipment-Grouping Strategies in Dynamic Chemical Plant Simulation", Proc 1982 Summ Simul Conf, July 1982.

Farrow LA and Edelson D (1974), "The Steady State Approximation : Fact or Fiction", Int J Chem Kinectics, Vol 6, p 787, 1974.

Feng A, Holland CD & Gallun SE (1984), "Development and Comparison of a Generalised Semi-Implicit Runge-Kutta Method with Gear's Method for Systems of Coupled Differential and Algebraic Equations", Comput Chem Eng Vol 8, No 1, pp 51-59, 1984.

Field AJ, Maddams RP and Morton W (1985), "The Incorporation of Procedures in an Equation-Oriented Flowsheeting Environment", IChemE Symp Ser 92, p 341, 1985.

Finlayson BA (1980), "Nonlinear Analysis in Chemical Engineering", McGraw Hill Book Co, 1980.

Fletcher JP and Ogbonda JE (1985), "The Development of a Dynamic Simulation Package for Chemical Processes", Design '85, Conference on Computers and Design in

the Process Industries, Organised by Midland Branch of IChemE, University of Aston in Birmingham, September 11-12, 1985.

Fletcher JP and Ogbonda JE (1987), "A Modular Equation-Oriented Approach to Dynamic Simulation of Chemical Processes", Proceedings of XVIII Congress on The Use of Computers in Chemical Engineering, CEF '87, Giardini Naxos, Sicily, Italy, April 26-30, 1987.

Franks RGE (1972), "Modelling and Simulation in Chemical Engineering", John Wiley Interscience, NY, 1972.

Franks RGE (1982), "DYFLO Update: DYFLO2" 1982 Summer Computer Simulation conference, Denver, Colorado (July 19-21, 1982), pp 507-513.

Fredenslund A, Gmehling J, Rasmussen P (1977), "Vapour-Liquid Equilibria using UNIFAC", Elsevier, Amsterdam, 1977.

Gallun SE and Holland CD (1980), "A Modification of Broydon's Method for the Solution of Sparse Systems - with Application to Distillation Problems Described by Non-ideal Thermodynamic Functions", Comput Chem Eng, Vol 4, pp 93-99, 1980.

Gallun SE and Holland CD (1982), "Gear's Procedure for the Simultaneous Solution of Differential and Algebraic Equations with Application to Unsteady State Distillation Problems", Comput Chem Eng Vol 6, No 3, pp 231-244, 1982.

Gear CW (1971a), "Numerical Initial Value Problems in ODEs", Prentice-Hall, Englewood Cliffs, NJ, 1971.

Gear CW (1971b), "DIFSUB for the Solution of ODEs", Commun ACM Vol 14, No 3, pp 185-190, 1971.

Gear CW (1971c), "Simultaneous Numerical Solution of Differential-Algebraic Equations", IEEE Trans on Circuit Theory, CT-18, No 1, pp 89-95, 1971.

Gear CW and Petzold L (1984), "ODE Methods for the Solution of Differential-Algebraic Systems" SIAM J NUMER ANAL, Vol 21 (4), pp 716-728, 1984.

Gear CW and Tu K (1974), "The Effect of Variable Mesh Size on the Stability of Multistep Methods", SIAM NUMER ANAL, Vol 11, (1974), pp 1025-1043.

Gorczynski EW and Hutchison HP (1978) , "Towards a Quasi-linear Process Simulator I: Fundamental Ideas", Comput Chem Eng, 2, p189, 1978.

Gorczynski EW, Hutchinson HP and Wajih ARM (1979), "Development of a Modularly Organized Equation-Oriented Process Simulator", Comput Chem Eng, Vol 3, pp353-356.

Hartzog DG and Davidson DL (1982) "Application of ACSL to Process Plant Simulation", Proceedings 1982 Summer Simulation Conference, pp 498-503, Simulation Council Inc, July 1982.

Heyt JW and Fairchild BT (1982), "Process Dynamic Simulations - A Potential Users Perspective", Proceed 1982, Summer Simulation Conf, pp 548-553, Simulation Councils Inc.

Hillesstad M (1986), "A Sequential Modular Approach to Dynamic Simulation of Chemical Engineering Systems", PhD Thesis, The University of Norway, 1986.

Hindmarsh AC (1981), "ODE Solvers for use with the Method of Lines", Advances in Computer Methods for Partial Differential Equations IV, R. Vichnevetsky and RS Stepleman, eds, IMACS, New Brunswick, N.J., pp 312-316, 1981.

Hindmarsh AC (1983), "ODEPACK - A Systematized Collection of ODE Solvers" Scientific Computing, IMACS Transactions on Scientific Computation, edited by Stepleman RS, 1983, pp 55-64, 1983.

Hlavacek V (1977), "Analysis of a Complex Plant - Steady State and Transient Behaviour", Comput Chem Eng Vol 1, pp75-100, 1977.

Holland CD and Liapis AI (1983) "Computer Methods for Solving Dynamic Separation Problems", McGraw-Hill, NY.

Husain A (1986), "Chemical Process Simulation" Wiley Eastern Ltd, New Delhi, India.

Hutchinson HP, Jackson DJ and Morton W (1986) "The Development of an Equation-Oriented Flowsheet Simulation and Optimization Package I: The QUASLIN PROGRAM", Comput Chem Eng Vol 10, No 1, pp19-29, 1986

Jackson KR and Sacks Davis R (1980), "An Alternative Implementation of Variable Stepsize Multistep Formulas for Stiff ODEs", ACM Trans Math Software, Vol 6, No 3, pp 295-318, 1980.

Joglekar GS and Reklaitis GV (1984), "BOSS - A Simulator for Batch and Semi-Continuous Processes", Comput Chem Eng, Vol 8, No 6, pp 315-327, 1984.

Kocak CM (1980), "Dynamic Simulation of Chemical Plant", PhD thesis, University of Aston in Birmingham, England.

Korn DA and Wait JV (1978), "Digital Continuous Systems Simulation", Prentice-Hall Inc, Inglewood Cliffs, NJ, USA, 1978.

Kubicek M (1976), "Algorithm 502 - Dependence of Solution of Nonlinear Systems on a Parameter", ACM Trans Math Software, Vol 2, pp 98-107, 1976.

Kuru S (1981), "Dynamic Simulation with an Equation-Based Flowsheeting System", PhD Thesis, Carnegie Mellon University, Pittsburgh.

Kuru S and Westerberg AW (1985), "A Newton-Raphson Based Strategy for Exploiting Latency in Dynamic Simulation", Comput Chem Eng Vol 9 No 2, pp 175-182, 1985.

Leesley ME and Buchmann AP (1980), "Databases for Computer-aided Process Plant Design", Comput Chem Eng, Vol 4, pp 79-83, 1980.

Leis JR and Krammer MA (1985), 'Sensitivity Analysis of Systems of Differential and Algebraic Equations", Comput Chem Eng Vol 9, No 1, pp 93-96.

Liang Wen-Chien (1985), "An Interactive Dynamic Flowsheet Simulation Program", PhD Thesis, Lehigh University.

Locke MH (1981), "A CAD Tool which Accommodates an Evolutionary Strategy in Engineering Design Calculations", PhD Thesis, Carnegie-Mellon University, Pittsburgh,

PA.

Lucia A (1983), "A Note on the Broyden-Householder Update", Comput Chem Eng Vol 7, No 2, pp 129-131, 1983.

Mahalec V, Kluzik H and Evans LB (1979) "Simultaneous Modular Algorithm for Steady State Flowsheet Simulation and Design", Presented at the 12th European Symposium on Computer Applications in Chemical Engineering, Montreux, Switzerland, April 8-11, 1979.

Michelsen ML (1976), "An Efficient General Purpose Method for the Integration of Stiff ODEs", AIChE J, Vol 22, NO , pp594-597.

Mitchell EEL (1978), "Advanced Continuous Simulation Language (ACSL)", in Numerical Methods for Differential Equations and Simulation, Bennett AW and Vichnevetsky (eds), IMACS, North Holland Publishing Company, 1978, pp 139-146.

Mitchell AR and Griffiths DF (1980), "The Finite Difference Method in Partial Differential Equations", Wiley, Chichester.

Motard RL, Shacham M and Rosen EM (1975) , "Steady State Chemical Process Simulation", AIChE J, 21, p417.

Paloshi JR (1982), "The Numerical Solution of Nonlinear Equations Representing Chemical Processes", PhD thesis, University of London, 1982.

Pantelides CC, Gritsis D, Morison KR and Sargent RWH (1987), "The Mathematical Modelling of Transient Systems using Differential-Algebraic Equations", Proceedings of the XVIII Congress on The Use of Computers in Chemical Engineering, CEF '87, Giardini Naxos, Sicily, Italy, April 26-30, 1987.

Patterson GK and Rozsa RB (1980), "DYNSYL - A General Purpose Dynamic Simulator for Chemical Processes", Comput Chem Eng 4, pp1-20, 1980.

Perkins JD (1979), "Efficient Solution of Design Problems using a Sequential Modular Flowsheeting Programme", Presented at the 12th European Symposium on Computer Application in Chemical Engineering, Montreuz, Switzerland, April 8-11, 1979.

Perkins JD (1984), "Equation-Oriented Flowsheeting" in Proceedings of the Second International Conference at Foundations of Computer Aided Process Design", (Eds JAW Westerberg and HH Chen), CACHE.

Perkins JD and Sargent RWH (1982), "SPEED-UP - A Computer Program for Steady State and Dynamic Simulation and Design of Chemical Processes", AIChE Symp Ser No 214, Vol 78, AIChE, NY.

Petzold L (1982), "Differential-Algebraic Equations are not ODEs", SIAM J Sci Stat Comput Vol 3, No 3, Sept 1982, pp 367-384.

Petzold L (1983), "A Description of DASSL: A Differential-Algebraic System Solver". Scientific Computing with R Stepleman et al (Eds), IMACS/North Holland Publ Co, 1983, pp 65-68.

Petzold L and Hindmarsh AC (1982), "LSODAR - Livermore Solver for ODEs with Automatic Switching Method for Stiff and Nonstiff Problems and with Root Finding Jan 27 Version", Lawrence Livermore National Laboratories, Livermore, CA 94550, 1982.

Powell MJD (1970), "A Hybrid Method for Nonlinear Equations", in "Numerical Methods for Nonlinear Algebraic Equations", Rabinowitz P, ed, Gordon and Breach, London, 1970.

Prausnitz JM, Anderson TF, Grens EA, Eckert CA, Hsieh R and O'Connell JP (1980), "Computer Calculations for Multicomponent VLE and LLE", Prentice-Hall Inc, Inglewood Cliffs, NJ, 1980.

Pritsker AAB and Hurst NR (1973), "GASP IV - A Combined Continuous-Discrete Fortran-Based Simulation Language", Simulation, Sept, pp 65-71, 1973.

Proctor SI (1983), "The FLOWTRAN Simulation System", Chem Eng Prog, pp 49-53, June 1983.

Rheinboldt WC (1984), "Differential Algebraic Systems as Differential Equations on Manifolds", Math Comput Vol 43, No 168, Oct 1984, pp 473-482.

Rheinboldt WC & Burkardt JV (1983), "A Locally Parametrized Continuation Process". ACM Trans Math Software Vol 9, pp 215-235.

Rosen EM (1980), "Steady State Chemical Process Simulation : A State-of-the-art Review" in Computer Applications to Chemical Engineering, RG Squires and GV Reklaitis (Eds), ACS Symp Ser, Vol 124, (3), 1980.

Rubner-Peterson T (1973), "An Efficient Algorithm using Backward Time-Scaled Differences for Solving Stiff Differential-Algebraic Systems", Institute of Circuit Theory and Telecommunication, Technical University of Denmark, 2800 Lyngby, 1973.

Sargent RWH (1981), "A Review of Methods for Solving Nonlinear Algebraic Equations", in RSH Mah and WD Seider (eds), Foundations of Computer-Aided Chemical Process Design, Vol 1, pp27-76, Engineering Foundation NY.

Sargent RWH and Westerberg SE (1964), "SPEED-UP in Chemical Engineering Design", Trans Inst Chem Engrs, 42, TI90, 1964.

Seader JD (1985), "Computer Modelling of Chemical Processes", AIChE Monograph Series, Vol 81, No 15, 1985.

Shacham M, Macchietto S, Stutzman LF and Babcock P (1982), "Equation-Oriented Approach to Process Flowsheeting", Comput Chem Eng Vol 6, No 2, pp 79-95,1982.

Shampine LF and Gordon MK (1975), "Computer Solution of Ordinary Differential Equations", WH Freeman and Co, San Fransisco, 1975.

Sincovec RF, Erisman AM, Yip EL and Epton MA (1981), "Analysis of Descriptor Systems using Numerical Algorithms", IEEE Trans, Automatic Control, AC-26 (1981), pp 139-147

Smith GJ and Morton W (1987), "Dynamic Simulation using an Equation-Orientated Flowsheeting Package", Proceedings of the XVIII Congress on The Use of Computers in Chemical Engineering, CEF '87, Giardini, Naxos, Sicily, Italy, April 26-30, 1987.

Soderlind G (1980), "DASP3 - A Program for the Numerical Integration of Partitioned Stiff ODEs and Differential/Algebraic Systems", The Royal Institute of Technology, Stockholm, Sweden, Dept of Numerical Analysis and Computing Science, TRITA-NA-8008, 1980.

Sood MK, Reklaitis GV and Woods JM (1979), "Solution of Material Balances for Flowsheets Modelled with Elementary Modules", AIChE J, Vol 25, p209, 1979.

Stadtherr MA & Hilton CM (1982a), "On Efficient Solution of Large Scale Newton-Raphson Based Flowsheeting Problems in Limited Core", Comput Chem Engng, Vol 6, p 115, 1982.

Stadtherr MA and Hilton M (1982b), "Development of a New Equation-Based Process Flowsheeting Systems: Numerical Studies", in Selected Topics on Computer-Aided Process Design and Analysis (eds RS Mah and GV Reklaitis), AIChE Symp Ser (1982).

Stadtherr MA and Wood ES (1984), "Sparse Matrix Methods for Equation-Based Chemical Process Flowsheeting I - Reordering Phase", Comput Chem Eng Vol 8, pp9-33.

Starner JW (1976), "A Numerical Algorithm for the Solving of Implicit Algebraic-Differential Systems of Equations", Tech Rep 318, Dept of Maths and Statistics, University of New Mexico, May 1976

Thambynayagom RKM, Wood RK and Winter P (1981), "DPS - An Engineer's Tool for Dynamic Process Analysis", The Chemical Engineer, 58-65.

Tsubaki M and Motard RL (1979), "Data Based Process Simulation", Comput Chem Eng Vol 3, pp 421-434, 1979.

Van Meulebrouk MGG, Swenker AG & de Leeuw den Bouter JA (1982), "Using TISFLO-II for the Optimization of Utility Generation and Supply for a Chemical Complex", IChemE Symp Ser, 74 (7), 1982.

Wait JV and Clarke D (1978), "DARE-P User Manual, Version 4.2", University of Arizona, College of Engineering, July, 1978.

Westerberg AW (1980), "Optimisation in Computer-Aided Design" in Proceedings of the Foundation of Computer-Aided Chemical Process Design", New Hampshire, July 6-11, 1980, pp 149-183.

Westerberg AW (1981), "Computer-Aided Design Tools in Chemical Engineering Process Design", Proceedings of the IEEE, Vol 69, No 10, pp 1232-1239.

Westerberg AW & Benjamin DR (1985), "Thoughts on a Future Equation-Oriented Flowsheeting System", Comput Chem Eng, Vol 9, No 2, pp 517-526, 1985.

Westerberg AW and Director SW (1978), "A Modified Least Squares Algorithm for Solving Sparse NxN Sets of Nonlinear Equations", Comput Chem Eng, 2, p77 (1978).

Westerberg AW, Hutchison H P, Motard R L & Winter P (1979), "Process Flowsheeting", Cambridge University Press, Cambridge, 1979.

Wood RK, Thambynayagom RKM, Noble RG and Sebastian DJ (1984), "DPS - A Digital Simulation Language for the Process Industries", Simulation, pp221-233, May, 1984.